Кем Кейнер, Джеймс Бах, Брет Петтикорд

Тестирование программного обеспечения

контекстно ориентированный подход

Десятилетия опыта тестирования
сжаты до самых важных уроков

Lessons Learned in Software Testing

A Context-Driven Approach

Cem Kaner
James Bach
Bret Pettichord

Wiley Computer Publishing

John Wiley & Sons, Inc.
NEW YORK • CHICHESTER • WEINHEIM • BRISBANE • SINGAPORE • TORONTO

Тестирование программного обеспечения

контекстно ориентированный подход

Кем Кейнер
Джеймс Бах
Брет Петтикорд

ПИТЕР®
Санкт-Петербург • Москва • Минск
2025

Выпущено
при поддержке
КРОК

ББК 32.973.2-018-07
УДК 004.415.53
К33

Кейнер Кем, Бах Джеймс, Петтикорд Брет

К33 Тестирование программного обеспечения: контекстно ориентированный подход. — СПб.: Питер, 2025. — 352 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-2165-6

Хорошими тестировщиками не рождаются — ими становятся благодаря упорному труду и постоянному общению. На этом пути таится множество ловушек, способных сорвать самые смелые планы и привести к отставанию проектов от графика.

Кем Кейнер, Джеймс Бах и Брет Петтикорд очень хорошо об этом знают. За их плечами более 50 лет опыта, и они понимают, что необходимо для достижения успеха в тестировании. Они собрали 293 проверенных совета, которые вы можете использовать в своих проектах. Каждый урок начинается с утверждения, относящегося к тестированию программного обеспечения, за которым следует объяснение или пример, показывающий, как, когда и почему применяется этот урок.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018-07
УДК 004.415.53

ISBN 978-0471081128 англ.

ISBN 978-5-4461-2165-6

Краткое содержание

Оглавление

Брайану Марику и Сэму Гакенхаймеру,
подсказавшим идею этой книги.

Дэйву Гельперину, поверившему в нас
и создавшему сообщество.

Джерри Вайнбергу, жизнь и работа
которого воплощают самые высокие
идеалы эксперта-тестировщика.

Памяти Анны Эллисон, коллеги и друга,
30 сентября 1952 г. — 11 сентября 2001 г.

Отзывы о книге

Откройте эту книгу на любой странице, почитайте пару минут — и у вас в копилке появится новый совет. Станьте профи по планированию, управлению и стратегиям тестирования.

Джоанна Ротман, компания Rothman Consulting Group, Inc.

Как бы вы ни тестировали ПО: сами или отдавая на аутсорс, прочитайте эту книгу. Каждая ее страница содержит полезные советы по решению ежедневных практических задач.

Сэм Гакенхаймер, старший директор по технологиям автоматизированного тестирования, корпорация Rational Software

Определенно, эта книга стоит того, чтобы прочитать ее и держать под рукой. Она толковая, имеет глубокий практический смысл и заставляет задуматься.

Росс Коллард, Collard & Company

Три выдающихся профессионала в области тестирования написали четко сформулированную и заставляющую задуматься книгу. В ней предложен особый взгляд на тестирование и управление тестовыми проектами.

Рекс Блэк, автор книг Managing the Testing Process и Critical Testing Processes

Эту книгу сообщество тестировщиков искало долгие годы. Ее следует прочесть всем руководителям и инженерам в сфере тестирования.

Джордж Хэмблен-младший, директор по обеспечению качества ПО крупной финансовой компании

Это не учебник. Это гораздо лучше. В книге описаны реальные примеры, обсуждаемые в рамках исследований. Я рад, что так много советов на тему тестирования собрано в одном издании. Думаю, оно породит множество замечательных дискуссий.

Стив Толман, менеджер по качеству ПО, PowerQuest

Уроки в этой книге представляют собой бесценные практические советы по тестированию ПО, полученные от ведущих экспертов. Если вы тестируете ПО или работаете в команде с тестировщиками, то эта книга — отличный источник знаний.

Алан Мирвольд

Четко и лаконично. Книга внесла ясность во многие ситуации из моего учебного опыта и породила множество новых идей.

Фран Маккейн, менеджер по тестированию ПО, компания Hewlett-Packard

Рецензирование этой книги стало для меня одним из самых ярких впечатлений. Я настойчиво рекомендую приобрести ее и держать под рукой всем специалистам по тестированию.

Ханс Бувальда, автор книги Integrated Test Design and Automation

Это золотой самородок, возникший как результат многолетнего практического опыта. Одна только глава, посвященная автоматизации тестирования, способна заменить все известные мне книги по данной теме. Глава о техниках содержит мощные идеи, изложенные простым языком!

Дуг Хоффман, консультант
в компании Software Quality Methods, LLC

Эту книгу обязательно должны прочесть начинающие специалисты, занятые поисками проверенных и рабочих решений, а также практикующие менеджеры по тестированию, изучающие новые возможности усовершенствования рабочего процесса для своей организации.

Крис Денардис, руководитель отдела разработки ПО
компании Rockwell Automation

...Предлагает бесценную коллекцию реальных практик, основанных на многолетнем опыте, которым поделились авторы и их коллеги. Абсолютно необходимая книга для всех, кто серьезно интересуется тестированием ПО.

Хунг К. Нгуен, президент и генеральный директор компании LogiGear Corporation, автор книги Testing Applications on the Web

Уроки просты и лаконичны — как раз то, что нужно для планирования тестирований поздним вечером. В других книгах много теории, и они отлично подходят для учебы, а в этой много реальных примеров, практического смысла и непосредственной пользы.

Мэри Ромеро Суини, автор книги Visual Basic for Testers

Превосходная книга. У меня был опыт, аналогичный описанному здесь, но не было возможности извлечь уроки, находясь в центре проблемы.

Стеле Амланд, Amland Consulting, Норвегия

Предисловие

Представьте, что держите в руках бутылку 50-летнего портвейна. Есть правильный способ пить портвейн. Он не единственный, но большинство людей, которые наслаждаются портвейном долгие годы, следуют определенным рекомендациям, которые помогают им получить максимально приятный эффект от его употребления. Вот лишь некоторые из них.

- **Урок 1. Не пейте из горлышка.** Если у вас нет ни бокала, ни другой посуды, то налейте немного портвейна на ладонь и пейте с нее. Делая глоток, вы должны ощущать аромат портвейна. Дайте напитку растечься по языку. Не пейте портвейн большими глотками.
- **Урок 2. Не выпивайте всю бутылку сразу.** Если вы пьете из-за того, что вас мучает жажда, то отставьте портвейн в сторону и выпейте воды. Вы получите максимальное удовольствие от портвейна, только если будете пить его маленькими порциями.
- **Урок 3. Не портите портвейн.** Если кто-нибудь посоветует вам попробовать коктейль из апельсинового сока, морской воды и портвейна, вежливо откажитесь. С широкой улыбкой скажите: «Я с удовольствием выпью просто бокал портвейна».
- **Урок 4. Не жадничайте.** Если вы будете придерживать напиток для себя, то никогда не узнаете, какое это удовольствие — вести приятную беседу, потягивая портвейн. Лучше всего пить его с друзьями, которые не прочь пропустить с вами бокальчик. Помните, что у них тоже где-то припрятана бутылка.

Вы держите в руках не бутылку портвейна, а бесценную книгу о тестировании ПО. Она доходила до совершенства на протяжении 50 лет работы авторов. Если портвейн предназначен для ваших вкусовых рецепторов, то эта книга — для вашего мозга. Думаю, что все остальные различия покажутся вам незначительными. Я ознакомился с данной книгой и хочу дать вам несколько советов, которые помогут получить максимум удовольствия от чтения.

- **Урок 1. Не пейте из горлышка.** Принесите свой бокал. То есть читайте эту книгу сквозь призму своего опыта разработки и тестирования ПО. Если вы никогда не принимали участия в серьезной работе над ПО, то при чтении этой книги можете на какое-то время впасть в замешательство. Если у вас есть опыт работы, то оцените содержание книги в контексте собственных проектов.
- **Урок 2. Не выпивайте всю бутылку сразу.** Не читайте эту книгу за один присест. Прочитайте пару уроков, закройте книгу и подумайте над словами Кейнера, Баха и Петтикорда. Вы узнаете, что они называют свой подход к тестированию контекстно ориентированным. Контекст собственной работы знаете только вы. Вам предстоит самостоятельно определить соответствие между данным уроком и вашей конкретной работой.
- **Урок 3. Не портите портвейн.** Кто-то может захотеть составить список названий 293 уроков, содержащихся в данной книге. Пожалуйста, пусть это будете не вы. Основу книги составляют пояснения к каждому уроку. Может быть также, что кто-то другой немедленно попытается привести ее к стандарту ISO или подвести содержимое под модель CMM. Представляется название статьи: «Достижение 293-го уровня CMM с помощью книги “293 урока тестирования ПО”». Кошмар! Как поясняют авторы, «...мы не верим в “лучшие практики”. Мы считаем, что при определенных обстоятельствах одни техники более полезны, чем другие». Будучи написанными профессионалами, эти уроки представляют собой квинтэссенцию достижения состояния мастера-исследователя тестирования ПО.
- **Урок 4. Не жадничайте.** Если и существует книга, которую следует читать вместе с коллегами, то это именно она. Купите несколько экземпляров и раздайте всем тестировщикам и тем, кто ими руководит. Прочитайте пару уроков и обсудите их за чашкой кофе, обедом или даже портвейном! Читайте, размышляйте, наслаждайтесь.

Ваше здоровье!

Тим Листер,
17 августа 2001 г.,
lister@acm.org,
компания Atlantic Systems Guild, Inc., Нью-Йорк

Введение

Международный стандарт SWEBOK (Software Engineering Body of Knowledge; «Свод знаний по программной инженерии») предлагается в качестве подходящей основы для государственного лицензирования, регулирования деятельности инженеров-программистов и разработки университетских учебных программ по программной инженерии. Документ SWEBOK претендует на звание основанного на консенсусе. Можно ожидать, что такой документ будет нести в себе знания и мудрость (уроки), накопленные в данной отрасли.

Вот что в SWEBOK говорится об исследовательском тестировании:

> «Пожалуй, самой распространенной техникой остается свободное тестирование: тестировщик создает тесты, полагаясь на свое мастерство и интуицию ("исследовательское" тестирование), а также на личный опыт работы с аналогичными программами. Хотя рекомендуется использовать более систематический подход, свободное тестирование может быть полезным (но только если тестировщик действительно является экспертом!) для выявления особых тестов, которые нелегко "уловить" с помощью формализованных методик. Кроме того, следует напомнить, что показатели эффективности этой методики могут сильно варьироваться» (SWEBOK 0.95, 2001. P. 5–9).

Как в SWEBOK трактуется то, что, по признанию его составителей, является наиболее широко применяемой техникой в данной области? В документе нет ни слова о том, как правильно применять эту технику. Есть лишь заявление, что исследовательским тестированием должны заниматься только *настоящие эксперты*, что рекомендуются иные подходы, а также высказывается предположение, что другие, формализованные методики будут менее эффективны.

Ха!

Мы *не* утверждаем, что наша книга выражает общее мнение и представляет собой *свод знаний* в нашей области, но нам *есть что сказать* о наиболее распространенных практиках в этой сфере. Вместо того чтобы отвергать исследовательское тестирование, в этой книге мы показываем, как выглядит тестирование глазами людей, использующих эти (и многие другие) техники в стремлении добиться превосходного тестирования в реальных условиях.

ДОБРО ПОЖАЛОВАТЬ

Эта книга рассказывает о разработке ПО в том виде, в котором мы пережили этот процесс. Суммарно наш опыт разработки составляет 50–60 лет (в зависимости от того, как считать).

Эта книга не о том, какой могла бы быть программная инженерия в идеальном и контролируемом мире. Мы пишем о *своем* опыте работы в реальном мире.

В нашем мире команды разработчиков ПО часто работают в условиях жестких дедлайнов, выясняя, что нужно сделать, и одновременно изучая способы реализации этих замыслов. Иногда их подходы более формальны, иногда менее. Это зависит от целого ряда обстоятельств.

Мы придерживаемся *контекстно ориентированного подхода* в тестировании ПО. Мы предполагаем, что метод, прекрасно работающий в одних условиях, не будет работать в других. Вместо того чтобы говорить о лучших практиках, мы говорим о практиках, подходящих для конкретного контекста.

Мы стараемся выбирать практики с учетом конкретных обстоятельств, чтобы добиться отличных результатов. Мы не думаем, что, взяв проект в свои руки, добьемся высоких результатов тестирования, и не станем указывать руководителю проекта (или исполнителям), как должны вести проект настоящие профессионалы. Нам кажется, что в тестировании нельзя добиться высоких результатов, запугивая программистов или заискивая перед ними. Мы не думаем, что отличное тестирование можно провести, заполняя тысячи листов бумаги (или электронных документов) и тратя время других сотрудников на ненужные процессы.

У нас нет ни политического, ни бюрократического, ни формально-методического рецепта отличного тестирования.

Такого рецепта просто не существует!

Мы считаем, что отличное тестирование включает в себя умелую техническую работу (поиск дефектов) и четкую, аргументированную коммуникацию.

Квалифицированный поиск всегда носит исследовательский характер. Тестировщикам нужно выполнить бесконечное количество тестов, при этом времени, за которое можно провести хотя бы какую-то их часть, ничтожно мало. На каждое тестирование, каждый документ, который мы пишем, каждое совещание, на котором мы присутствуем, тратится время, которое можно было бы потратить на выполнение тестов, способных помочь выявить ключевой дефект. Столкнувшись с этим ограничением, мы оптимизируем процессы тестирования таким образом, чтобы в них использовались наши постоянно растущие знания о продукте, его рынке, областях применения и слабых сторонах. То, что мы узнаём сегодня, отразится в более мощных тестах завтра.

Даже если:

- у продукта хорошая спецификация и…
- она точно отражает документ с требованиями и…
- он точно отражает реальные потребности стейкхолдеров (заинтересованных сторон) продукта —

(вы *когда-нибудь* участвовали в проекте, где все эти «даже если» были правдой?), мы все равно узнаём много нового о том, как тестировать продукт, пока делаем это. В частности, по мере обнаружения ошибок мы узнаём, как конкретная команда программистов может ошибаться. В спецификациях рассказывается о том, как должна работать программа, если она написана правильно. В них не говорится, какие ошибки следует предвидеть и как разрабатывать тесты для их обнаружения. Выполняя эту задачу (а она является для нас ключевой), мы улучшаем свои навыки за то время, пока длится проект. И так происходит в каждом проекте.

Когда мы проводим тестирование и наш мозг активно работает, наша работа носит исследовательский характер. Независимо от того, как это выглядит со стороны.

ДЛЯ КОГО ЭТА КНИГА

Книга предназначена для тех, кто занимается тестированием ПО, руководит тестировщиками, а также для тех, кому приходится работать со специалистами в этой области в своих проектах по разработке ПО.

В этой книге, говоря «вы», мы обращаемся к нашему основному читателю — к людям, которые работают в компании уже несколько лет и, возможно, недавно были переведены на руководящую должность. Надеемся, в уроках вы отметите много общего со своим опытом, получите новые знания, найдете такие уроки, которые сочтете полезным процитировать вашему руководителю. Мы также надеемся, что вам очень понравятся некоторые из наших утверждений и вы распечатаете их и повесите на стену в своем кабинете и, возможно, отреагируете хотя бы на одно утверждение настолько сильно, что прикрепите его копию на середину своей доски для дартса. (Мы хотим заставить вас размышлять, а не только добиться согласия.)

У тестировщиков-новичков (и у тех из вас, кто только устраивается на эту работу) ощущение, что вы уже сталкивались с тем, о чем мы пишем, будет возникать не слишком часто. Для вас эта книга может послужить неким первым толчком и предостережением, давая представление о том, с какими проблемами сталкиваются тестировщики.

Подсказка. Если вы новичок и ищете книгу, которая помогла бы научиться тестированию и подготовиться к собеседованию, то наша книга вам не подойдет. Если у вас нет других подобных книг, то обратите самое пристальное внимание на главы 3 «Техники тест-дизайна» и 10 «Ваша карьера в области тестирования программного обеспечения». А лучше всего прочесть первые пять глав книги *Testing Computer Software*[1].

Программисты, руководители проектов и другие специалисты, которым приходится работать с тестировщиками, найдут в этой книге полезные идеи, позволяющие определить ожидания, которые возлагаются на команду тестирования. Мы надеемся, что это поможет вам оценить работу ее участников и обсудить с ними проблемы, если вы не согласны с их политикой или считаете, что они неразумно тратят свое время.

---

[1] *Канер С. и др.* Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений.

О ЧЕМ ЭТА КНИГА

За годы работы мы узнали много полезных практик и способов оценки ситуаций. Наши выводы основываются на опыте. В книге мы обобщаем его значительную часть, преподнося выводы в виде коротких удобочитаемых описаний нескольких сотен *уроков*.

Мы приняли несколько критериев добавления того или иного урока в книгу.

- Урок должен быть полезным или давать понимание темы.
- Урок должен пройти 90-минутный тест на самостоятельность мышления. Иначе говоря, урок не стоит добавлять в книгу, если его смог придумать практически каждый, кто думал о тестировании в течение 90 минут, не отвлекаясь ни на что другое.
- Урок должен быть основан на нашем реальном опыте. Хотя бы один из нас (а лучше все трое) должен был успешно применять наши рекомендации. По крайней мере двое из нас должны были обжечься, пытаясь следовать практике, которую мы критикуем. (*Примечание.* Иногда мы приходим к разным выводам, основываясь на разном опыте. Иногда мы выбираем не одну, а две точки зрения. Даже если представлена только одна, нельзя считать, что все трое полностью с ней согласны, — в случае разногласий мы, скорее всего, отдадим предпочтение тому из нас троих, кто имеет наиболее богатый опыт в данной ситуации.)
- Уроки должны гармонировать с опытом наших коллег. Мы собирали подробные отчеты о нем на семинарах по тестированию ПО, проводимых в Лос-Альтосе, круглых столах для менеджеров по тестированию ПО, семинарах по эвристическим и исследовательским методам, семинарах по автоматизации тестирования, проводимых в Остине, семинарах по паттернам тестирования ПО, семинарах по автоматизированному тестированию на основе моделей, на десятках конференций по тестированию ПО и на менее формальных тренингах (например, в ежегодных лагерях консультантов в Крестед-Бьютте).
- Урок должен быть кратким и понятным, но при этом легким для усвоения.
- Урок может быть более длинным, но лишь настолько, чтобы можно было объяснить, как сделать что-то, или предоставить полезный инструмент. Длинные описания и подробная справочная информация приводятся в учебниках.

- Уроки должны быть самодостаточными. Вы должны иметь возможность начать чтение с любого урока.
- В совокупности уроки должны дать вам представление о том, как мы проводим тестирование, а также об образе мышления и логике тестировщиков.

ЧЕГО В ЭТОЙ КНИГЕ НЕТ

Книга не является подробным руководством по тестированию ПО.

Ее нельзя считать сборником абсолютно верных уроков. Это *наши* уроки, основанные на *нашем* опыте. Мы считаем, что они применимы в широком смысле, но некоторые уроки, оказавшиеся полезными и важными в нашей карьере, могут не подойти вам. Всегда следует опираться на собственные суждения. Универсальность нашей книги ограничивает тот факт, что мы больше занимались разработкой ПО для массового рынка и контрактного ПО, чем программным обеспечением, предназначенным для внутреннего использования. Наш опыт работы с жизненно важным ПО и встраиваемым ПО ограничен.

Кроме того, книга не является и сборником лучших практик. На самом деле мы не верим в «лучшие практики». Мы считаем, что при определенных обстоятельствах одни техники более полезны, чем другие. Мы обеспокоены тем, что многое из того, что продается в качестве лучших практик, продвигается (и применяется), не подвергаясь критичному осмыслению, в неподходящих ситуациях.

СТРУКТУРА ИЗДАНИЯ

Мы написали книгу таким образом, чтобы ее можно было читать, начиная с любой страницы, или пролистать, а не пытаться читать ее строго от начала до конца. В какой-то момент (надеемся) вы обнаружите жемчужину, идею, которая вам очень понравится. Мы не можем рекомендовать вам слепо применять эту идею, не подвергая ее критике. (Наши уроки — это не «лучшие практики».) Вместо этого мы призываем вас оценить урок с точки зрения его соответствия вашей ситуации.

Далее приведены несколько вопросов, которые помогут вам провести такую оценку.

- При каких обстоятельствах этот урок можно было бы применить в вашей компании?

- При каких обстоятельствах урок не будет работать?
- Пробовал ли кто-нибудь из ваших знакомых что-нибудь подобное этому уроку? И что произошло? Почему? Чем ваш текущий проект отличается от проекта этого человека? Значимо ли это различие?
- Кто с наибольшей вероятностью выиграет, попытавшись применить этот урок?
- Кто с наибольшей вероятностью окажется в невыгодном положении в результате попытки применения урока?
- Что вы узнаете, попытавшись применить этот урок?
- Пробуя что-то новое, вы рискуете. Имеет ли смысл проверить применимость урока в вашей компании, проведя пилотное исследование — создав ситуацию для опробования новых знаний, не принимая на себя обязательств, или когда риск минимален? Можно ли провести пилотное исследование в вашей компании в рамках создания текущего проекта (или в ходе его обслуживания)?
- Как оценить эффективность применения этого урока? Если решение сработает, то как узнать, чем был обусловлен успех: ценностью самого урока, сохраняющейся в процессе его дальнейшего применения, или вашим энтузиазмом при опробовании идеи?
- Что хорошего или плохого может произойти при попытке применить этот урок?
- Что хорошего или плохого может произойти с другим заинтересованным участником, например с пользователем или другим членом команды разработчиков?
- Что делать, если ключевая фигура в вашей компании не согласна с решением в уроке, которому вы хотите следовать? Как можно преодолеть такого рода возражения и затем продать это решение данному человеку?

НАДЕЕМСЯ, ЭТА КНИГА ВЫЗОВЕТ ДИСКУССИИ И ОБСУЖДЕНИЯ

В книге проводится резкий контраст между нашими и некоторыми другими взглядами. Мы считаем, что это поможет разжечь дискуссии, которые должны вестись в данной области. По нашему мнению, общепринятой парадигмы в тестировании ПО или в программной инженерии в целом не существует. Именно поэтому мы не относимся к числу тех, кто выступает за государственное лицензирование и регулирование деятельности инженеров-программистов после стандартизации *свода*

знаний. По нашему опыту, существуют совершенно разные, заслуживающие доверия мнения о том, как лучше выполнять работу.

Хотелось бы внести ясность в этот вопрос. Мы часто критикуем работу людей, которых очень уважаем. Во многих случаях мы ссылаемся на работы, выполненные нашими хорошими друзьями. Не путайте критику идеи с критикой ее сторонника или автора.

Мы считаем, что сравнение и противопоставление разных точек зрения пойдет на пользу сфере разработки. В этой области очень важно вести дискуссии о наших методах. Мы выступаем за дальнейшее развитие квалифицированной практики в рамках каждого из основных подходов. В конце концов, мы все узна́ем, при каких обстоятельствах различные подходы становятся наилучшими. А пока пусть процветает многообразие мнений.

НЕСКОЛЬКО ЗАМЕЧАНИЙ ПО ЛЕКСИКОНУ

Ниже приведены некоторые ключевые слова, встречающиеся в этой книге, и то, что мы в них вкладываем.

Мы. Авторы.

Вы. Читатель.

Сбой — это ошибка, оплошность в проекте или реализации программы.

В нашем понимании *ошибка* — это сбой.

Отказ — неправильное поведение программы, возникающее в результате сбоя.

Отказы происходят при определенных *условиях*. Например, при попытке деления на ноль произойдет сбой программы. В данном случае он кроется в коде, разрешающем деление на ноль. Отказ — это нарушение работы. Но он произойдет только тогда, когда критическая переменная при делении будет иметь значение, равное нулю. Эта переменная с нулевым значением представляет собой критическое условие, которое должно быть выполнено, чтобы произошел отказ.

Симптом похож на отказ, но является менее серьезной проблемой. Например, если в программе произошла утечка памяти, то ее выполнение может начать замедляться задолго до того, как произойдет сбой и будет выдана ошибка нехватки памяти — out-of-memory. Замедление

выполнения программы представляет собой симптом глубинной проблемы (нехватки памяти), которая не очевидна.

Баг — всеобъемлющий термин. Он может быть связан с какими-либо неполадками в программном обеспечении. Тот, кто сообщает о возникновении бага, может описать сбой, отказ или ограничение программы, которое делает ее менее ценной для стейкхолдера.

Если мы определяем *качество* как ценность для какого-то человека (Weinberg, 1998, 2.i), то *баг-репорт* — это заявление о том, что какой-то человек считает продукт менее ценным из-за того, что в нем есть баг.

Слово *«дефект»* несет в себе юридический смысл. Оно означает, что с продуктом явно что-то не так. Некоторые компании не разрешают использовать слова «дефект» или «дефектный» в системах отслеживания багов или связанных с ними служебных записках.

Некоторые компании предпочитают вместо *«баг»* употреблять такие понятия, как *«аномалия»*, *«проблема»* или *«вопрос»*.

После того как вы сообщите об обнаружении бага, программисты (или совет по управлению изменениями) исправят его или примут решение не исправлять. Они помечают баг как *решенный* (исправлен, отложен, не воспроизводим, работает как задумано и т. д.).

Тестирование методом «черного ящика». Тестирование внешнего поведения программы путем ввода входных данных и изучения выходных. При типичной процедуре тестировщик не имеет представления о внутреннем устройстве кода, но хорошо знаком с явными и неявными требованиями к продукту, такими как способы использования программы, типы поступающих в нее данных, любые нормативные вопросы, связанные с проблемой, которую программа пытается или помогает решить, а также аппаратное и программное окружение для обеспечения работы данной программы.

Поведенческое тестирование. Тестирование внешнего поведения программы, аналогичное тестированию методом «черного ящика», но с использованием всего доступного тестировщику объема знаний о внутреннем устройстве программы.

Функциональное тестирование. Тестирование методом «черного ящика» или поведенческое.

Тестирование методом «белого ящика», или «прозрачного ящика». Тестирование с использованием знаний о внутреннем устройстве программы.

Структурное тестирование. Тестирование методом «белого ящика», направленное на изучение внутренней структуры программы, например потока управления от одного решения или действия к другому.

Смоук-тестирование, или *верификационное тестирование сборки.* Стандартный набор тестов, применяемый к новой сборке. Тесты позволяют выявить фундаментальную нестабильность, отсутствие или поломку ключевых элементов. Если сборка не прошла эти тесты, то дальнейшее тестирование не проводится. Вместо этого возобновляется тестирование старой сборки или откладывается до создания новой.

Руководитель проекта. Человек, ответственный за своевременную поставку нужного продукта в рамках бюджета. В некоторых компаниях работу, которую, по нашему мнению, делает руководитель проекта, поручают руководителю группы проектов и руководителю разработки.

MaxInt. Наибольшее целое число, возможное на платформе пользователя или языке разработки программиста. Число, превышающее MaxInt, не может быть сохранено как целое.

Клиент. Некто, интересам кого вы обязаны служить. К клиентам, вероятно, в той или иной степени относятся все участники проекта, а также конечные пользователи продукта.

ОТ ИЗДАТЕЛЬСТВА

Ваши замечания, предложения и вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция). Мы будем рады узнать ваше мнение!

Выражаем огромную благодарность компании «КРОК» за помощь в работе над русскоязычным изданием книги и их вклад в повышение качества переводной литературы.

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

О научном редакторе русскоязычного издания

Захарова Анна — старший инженер-тестировщик. Более 15 лет работала ведущим инженером-тестировщиком на проектах арбитражной судебной системы РФ, судов общей юрисдикции, проектах РОССТАТ, проектах ФОМС, интеграционных проектах.

Благодарности

Выход этой книги был бы невозможен без поддержки и помощи многих людей. Мы благодарим Ленор Бах, Джона Баха, Бекки Фидлер, Лесли Смарт и Зака Петтикорда за поддержку, понимание и помощь, пока три сумасшедших человека не думали ни о чем, кроме своей книги. Благодарим Пэт Макги за помощь в проведении исследований в критический момент.

Нам помогли подробные и вдумчивые рецензии на первые черновики книги. Мы добавили в книгу несколько примеров и описаний альтернативных точек зрения, приведенных нашими рецензентами. Благодарим вас, Стеле Амланд, Рекс Блэк, Джеффри Блейберг, Ханс Бувальда, Росс Коллард, Лиза Криспин, Крис Денардис, Мардж Фаррелл, Дороти Грэм, Эрик Гриффин, Роки Гробер, Сэм Гакенхаймер, Джордж Хэмблен, Элизабет Хендриксон, Даг Хоффман, Кэти Иберл, Боб Джонсон, Карен Джонсон, Джинни Кейнер, Бартон Лейн, Пэт Макги, Фрэн Маккейн, Пэт Маккуэйд, Брайан Марик, Алан Мирволд, Хунг Нгуен, Ноэль Найман, Эрик Петерсен, Йоханна Ротман, Джейн Степак, Мелора Свобода, Мэри Ромеро Суини, Пол Шимковяк, Энди Тинкхэм, Стив Толман и Тамар Ярон.

Содержимое этой книги значительно улучшилось благодаря многочисленным обсуждениям на семинарах по автоматизации тестирования (Лос-Альтос), семинарах по эвристическим и исследовательским методам, круглом столе менеджеров по тестированию ПО, семинарах по автоматизации тестирования (Остин), семинарах по паттернам тестирования ПО, а также на многих других семинарах, конференциях, занятиях и площадках с большим количеством людей, вкладывающих всю душу в поиск лучших способов тестирования ПО. Мы благодарим каждого из вас.

Глава 1

Роль тестировщика

Что тестировщики должны делать в проекте? Как и многое другое в тестировании, ответ на данный вопрос может показаться очевидным или простым, но это не так.

Любая роль подразумевает взаимоотношения. Это означает, что вы не контролируете свою роль, но можете договориться, в чем она состоит. Люди ожидают от вас того, что может быть неразумным. Когда вас обвиняют в низком качестве продукта (а такое случается), обвиняющие, вероятно, неправильно понимают роль тестировщика. Возможно, они считают, что ваша работа — бить по продукту волшебным молотком качества, и думают, что вы бьете недостаточно сильно.

Четко представляя себе свою роль (обсудив и утвердив ее рамки), вы понимаете, на чем могут основываться ожидания в любой возникающей ситуации. Однако следование даже четкой и адекватной роли в тестировании требует больших усилий.

Вы — «свет фар» проекта

Проект — своего рода дорожное приключение. Некоторые проекты просты и рутинны, как поездка в магазин. Но реализация большинства проектов скорее напоминает ночное вождение грузовика по горному бездорожью. И здесь не обойтись без света фар. Как тестировщик, *вы освещаете путь*. Вы освещаете дорогу, чтобы программисты и руководители, как бы ни спорили по поводу дорожной карты, могли хотя бы видеть, где они находятся, куда едут и насколько они близки к краю обрыва. Конкретная цель команды тестировщиков меняется в зависимости от компании, но все нюансы и различия объединяет нечто общее.

Тестирование проводят для того, чтобы найти информацию. На ее основе принимают критические решения по проекту или продукту.

Ваши цели управляют тем, что вы делаете

Ваши цели могут зависеть от отрасли, компании, проекта или состава команды. Тестирование проектов сильно различается в разных компаниях. В процессе развития тестирования как ремесла возникла необходимость обсуждать практику тестирования, в ходе чего можно было бы преодолеть культурные и технические различия. Многие из них обусловлены разными целями команд тестирования. Например, в одних организациях тест-план — просто инструмент, помогающий тестировщикам. Он может быть написан на салфетке и все равно быть эффективным. Другие организации создают тест-план в виде продукта, который должен поставляться вместе с их программным обеспечением. Их тест-план соответствует строгим требованиям и инструкциям.

Некоторые из указанных ниже требований могут определить вашу цель. Чего ждут от вас?

- Быстро найти важные баги.
- Дать общую оценку качества продукта.
- Удостовериться, что продукт отвечает конкретным стандартам.
- Помочь клиентам улучшить качество и тестируемость продукта.
- Удостовериться, что процесс тестирования соответствует стандартам.
- Рассказать клиентам о тестировании и как работать с тестировщиками.
- Следовать конкретному набору методов и правил.
- Помочь предсказывать и контролировать расходы на техподдержку.
- Помочь клиентам улучшить их процессы.
- Выполнить свою работу с минимальными расходами, за минимальное время или с минимумом негативных последствий.
- Делать все, что требует конкретный клиент.

Если вы тратите время и силы на выполнение требований, которые не волнуют клиентов, то рискуете прослыть неактуальным и контрпродуктивным. Обсудите ваши цели с руководителем. Проясните их. Если вы не сможете прийти к согласию касательно целей, то вам не на что будет опереться в процессе работы.

Что вы должны делать, когда не знаете, что делать? Один из ответов — пересмотреть вашу цель. Она определяет суть проблемы, с которой вы столкнулись. Установив четкую цель тестирования, вы сможете защитить свою работу и определить, что будете делать дальше. Вы также сможете простыми словами объяснить окружающим, в чем заключается ваша роль. Если по каким-то причинам вы не можете выполнить свою задачу, то сразу же обратитесь к руководству.

Что вы должны делать, когда точно знаете, что делать? Время от времени возвращайтесь к вашим целям, чтобы убедиться в том, что в ходе четкого следования плану не погрузились в одну часть задачи тестирования и не забыли обо всех остальных.

Вы обслуживаете разных клиентов

Тестирование — это услуга. Прочувствуйте это. Услуга, которую вы предоставляете, жизненно важна. Услуга подразумевает наличие клиентов — людей, которых вы обслуживаете. Ваш успех измеряется в первую очередь тем, насколько удовлетворены желания и интересы клиентов. Это может быть не так уж сложно, если не считать того, что тестирование нужно *многим* клиентам. Каждому из них требуется что-то свое, и их запросы не всегда совпадают.

- *Руководитель проекта.* Руководители имеют право знать ваш процесс работы и влиять на него. Вы оказываете услугу руководителю проекта, сообщая ему о ходе работ, о важных проблемах и не становясь мешающим фактором. Прерогатива руководителя — управлять проектом. Ваша работа — сказать ему, что вы можете сделать, а что — нет и как повлияет на тестирование то или иное решение или условие проекта.
- *Программист.* Вы упрощаете работу программиста, создавая настолько ясные баг-репорты (bug reports, «отчеты о багах»), насколько это возможно. Стремитесь быть профессионалом в своем деле и разбираться в продукте, чтобы не тратить время программиста на ложные или поверхностные баг-репорты. Если вам это удастся, вы завоюете больший авторитет у разработчиков. В свою очередь, это приведет к тому, что вы получите их поддержку и сможете выстраивать продуктивное общение.
- *Технический писатель.* Как и вы, люди, которые составляют руководства и интерактивную справку, имеют неполную информацию

о продукте. Вы можете помочь им понять, как он работает на самом деле, и предупредить об ошибках в документации. Писатели тоже могут помочь вам. Изучая продукт и то, как его будут использовать люди, читающие документацию технические писатели узнают́ то, чего не знаете вы. Если у вас хорошие отношения с техническими писателями, то они будут сообщать вам о новых функциях, новых способах использования, слабых местах в ваших тест-планах и найденных багах. Некоторые из этих багов никогда не были бы найдены с помощью тест-планов.

- *Техническая поддержка.* Любая проблема в продукте становится бременем для людей, обеспечивающих техническую поддержку. Вы сообщаете команде поддержки о тех аспектах применения продукта, которые могут вызвать затруднения у пользователей. Если вы взаимодействуете с представителями техподдержки в процессе разработки, то иногда они могут помочь вам доказать, что баг необходимо исправить. В свою очередь, вы также можете предлагать свою помощь в изучении сложных проблем, возникающих в предметной области. Таким образом можно улучшить контакт между техподдержкой и клиентами.
- *Маркетинг.* Маркетологам нужно знать, не противоречит ли что-либо в продукте ключевым преимуществам, которые описываются клиентам. Баг, который кажется незначительным программистам, может оказаться критическим для маркетологов. Они могут понять, что из-за бага клиенту сложнее выполнить важную задачу. Кроме того, просматривая запланированные маркетинговые документы, вы можете помочь маркетологам составить точное представление о возможностях продукта.
- *Топ-менеджмент и стейкхолдеры.* Вы обслуживаете бизнес. Поэтому вы должны быть осторожными, чтобы выглядеть и вести себя как разумный человек, а не как фанатик качества. Особенно ближе к концу проекта выполняйте свою работу таким образом, чтобы учитывать краткосрочные и долгосрочные интересы компании. Составляйте отчеты о состоянии тестирования, используя четкие и понятные термины, чтобы руководители чувствовали, что им есть на что опереться при принятии решений.
- *Пользователи.* И разумеется, вы обслуживаете тех, кто будет пользоваться продуктом. Конечно, удовлетворение их запросов отвечает интересам вашего проекта. Но и то, что вы являетесь главным защитником интересов пользователей в проектной команде, приносит особое удовлетворение.

Этот список не отсортирован по приоритету, но в вашем проекте такая иерархия может быть, поэтому узнайте ее. Выясните цели своего проекта. Узнайте, кого вы обслуживаете. Это первый шаг к отличному тестированию.

То, что вы обнаружили, может быть багом с точки зрения людей, мнение которых имеет значение

В задачи вашей команды входит (или должно входить) информирование клиентов обо *всем*, что угрожает ценности продукта, определенной так, как ее понимают клиенты. Если вы считаете, что продукт не будет оценен по достоинству, даже если работает должным образом, то обязаны сообщить о своих опасениях. Если клиенты решат проигнорировать ваше сообщение, это их право.

Быстро находите критические баги

Вероятно, в ваши цели входит поиск критических багов (в отличие от незначительных), и он должен быть быстрым. Если да, то что это означает в контексте тестирования?

- Тестируйте *изменившиеся части* прежде неизменившихся. Исправления и обновления означают новые риски.
- Тестируйте *базовые функции* прежде вспомогательных. Проверяйте критические и популярные функции продукта. Тестируйте функции продукта, которые делают его таким, какой он есть.
- Тестируйте *возможности* прежде надежности. Проверяйте работоспособность *всего* функционала, прежде чем углубляться в изучение особенностей работы функций в различных условиях.
- Тестируйте *стандартные условия* прежде гипотетических. Рассмотрите популярные данные и сценарии использования.
- Тестируйте *распространенные угрозы* прежде малораспространенных. Проводите проверки на наиболее вероятные стрессовые и ошибочные ситуации.
- Тестируйте *проблемы, которые могут нанести больший вред*, прежде тех, последствия которых не слишком серьезны. Проверяйте модули продукта, которые в случае отказа могут причинить наибольший ущерб.

- Тестируйте *наиболее используемые области* прежде невостребованных. Проверяйте любые области и обсуждайте проблемы, которые интересны еще кому-либо в команде.

Кроме того, вы быстрее обнаружите важные проблемы, если будете больше знать о продукте, программном и аппаратном обеспечении, с которым он должен взаимодействовать, и о людях, которые будут его использовать. Хорошо изучите их.

Работайте вместе с программистами

Помощь программистам, вероятнее всего, является вашей ключевой задачей. Если вы тестируете функционал, который программисты создали только что или недавно, то ваша обратная связь поможет им быть более эффективными. Когда они поставляют продукт, вы его тестируете. Если они внесли какие-либо изменения — протестируйте их. Стремитесь к тому, чтобы цикл обратной связи был максимально коротким и быстрым. Пока программисты будут разбираться с багами, которые вы нашли, занимайтесь поиском других багов. Идеальная (для тестировщиков) ситуация — это когда программисты заняты исправлением найденных вами проблем так сильно, что именно они, а не вы являются узким местом проекта.

Спрашивайте обо всем, но не обязательно вслух

Можно тестировать, не задавая вопросов, но лучше так не делать. Вопросы — основа вашей роли в проекте. Если вы их не задаете, то тестируете бесцельно и механически. Однако учтите, что некоторые явные вопросы могут показаться провокационными и люди начнут защищаться.

Вопросы подобны сильному лекарству: их лучше давать малыми дозами или сочетать с другими видами общения. К счастью, ценность вопросов не уменьшается, если задавать их не вслух. Любой вопрос, который приходит вам в голову, может подтолкнуть ваши собственные мысли в нужном направлении, что приведет к важным открытиям.

Если однажды в процессе тестирования вы поймете, что у вас нет вопросов по продукту, — сделайте паузу.

Сосредоточьтесь на отказах, чтобы ваши клиенты могли фокусироваться на успехе

Тестировщик — единственный участник проекта, который не ориентирован непосредственно на успех. Другие участники что-нибудь создают или описывают уже созданное. А тестировщики ищут *негативные факторы*. Тестирование может быть депрессивной работой, сродни пародии на греческий миф: «На острове тестировщиков они были обречены вечно искать то, чего не могло быть и не должно было существовать, зная, что успех принесет несчастье богам».

Было бы ошибкой переопределить вашу цель *более позитивно* — что вы человек, который проверяет, работает ли программа. Даже если вам поручат «убедиться, что программа работает», расскажите своим клиентам, что такая проверка невозможна. Она стоит очень дорого. Даже проведя все возможные тесты, вы не сможете доказать, что продукт работает. Максимум, что вы сможете сказать: «В ходе проведенных тестов я не заметил, что продукт не работает». А вот обратная ситуация сулит большую экономию: всего одним-единственным тестом можно доказать, что продукт не работает.

Тестировщики фокусируются на отказах, поскольку это повышает шансы обнаружить их. Ищите ключевые проблемы в продукте, используя все свои творческие способности и навыки. Если вы их не найдете, то их нельзя будет исправить, и тогда пользователи могут обнаружить их за вас. Находя все возможные ошибки в продукте, вы помогаете членам команды проекта больше узнать об их навыках и рисках продукта, а также помогаете им улучшить продукт, сделать его более удобным для поддержки и, возможно, более успешным на рынке.

Вы никогда не найдете все баги в продукте

Ваша работа — искать значимые баги и сообщать о них. Но вы никогда не найдете все. Чтобы найти их все, вам придется искать баг везде, где он может быть, рассматривая каждую потенциально возможную ситуацию. Кроме того, вам понадобится надежный способ идентифицировать каждый тип возникшего бага. Если вы думаете, что можете это сделать, то у вас или очень простой продукт, или очень ограниченное воображение.

Вы должны выбрать, на что тратить свое время, зная, что не сможете сделать все.

Опасайтесь «завершенного» тестирования

Некоторые тестировщики соглашаются, что не могут знать наверняка, нашли ли они все баги в продукте, и тем не менее говорят о том, что означает «закончить тестирование». Слова «мне потребуется пять дней, чтобы протестировать это» можно интерпретировать так: вы считаете, что *полностью* протестируете эту часть продукта за пять календарных дней. А это, в свою очередь, может быть воспринято как то, что за пять дней вы найдете все баги. *Завершенность* чаще подразумевается, чем утверждается. В любом случае к этой концепции нужно относиться с большой осторожностью. Подумайте, что может означать завершенное тестирование, что именно завершено:

- поиск багов в продукте;
- изучение каждой части продукта;
- тестирование, которое, по вашему мнению, полезно и экономически выгодно на данный момент;
- выполнение целей проекта, которых вы достигали в меру своих способностей;
- выполнение согласованных тестов;
- проверка всего, что под силу протестировать человеку при данных обстоятельствах;
- все, что вы знали, как делать;
- ваша часть тестирования, вне зависимости от каких-либо других частей;
- широкое, но не глубокое тестирование продукта;
- один вид тестирования продукта;
- период времени, отведенный на тестирование.

Если вы позаботитесь о том, чтобы уточнить, что именно подразумеваете под словами «завершено», «закончено» или «сделано», вероятность того, что вас будут обвинять в плохом выполнении своей работы, уменьшится. А если все же обвинят, то вы сможете лучше защитить себя. Имейте в виду, что понятие «завершенность» невозможно окончательно определить в начале проекта. Вам придется пересматривать его по мере развития тестового проекта и появления новых тестовых задач.

Чтобы не было недопонимания того, что значит завершенность, поделитесь с клиентами некоторыми подробностями процесса тестирования.

Подведите итоги выполненного вами тестирования и объясните, почему оно имеет смысл, а также расскажите клиентам о других заслуживающих внимания тестах, которые вы не проводите, и о том, почему вы их *не* проводите.

Вы не гарантируете качество с помощью тестирования

Очень легко считать себя блюстителем качества. Но вы не повышаете качество и не снижаете его. Вы можете говорить, что «ломаете» продукт, однако на самом деле он пришел к вам уже «сломанным». Качество зависит от людей, которые создают продукт, и иногда это становится для них тяжелым бременем. Одна из главных ваших целей — помочь им эффективно справиться с этим бременем. Вы не сможете сделать это, если будете думать, что вы единственные в проекте, кто заботится о поставке качественного продукта.

Ваша команда может называться «Обеспечение качества». Воздерживайтесь от гордыни, не заноситесь. С помощью результатов тестов и баг-репортов вы предоставляете информацию, которая способствует обеспечению качества проекта, но само это обеспечение — результат усилий всей команды.

Никогда не будьте контролером!

Некоторые тестировщики хотят иметь право вето на выпуск продукта. Исполнение этого желания сродни наказанию. Проблема в том, что, когда тестировщики контролируют релиз, они также должны нести полную ответственность за качество продукта. Остальные члены команды немного (а может быть, и очень сильно) расслабятся. Если какой-либо баг ускользнет от внимания тестировщиков, да еще в самом начале проекта, то остальная команда может (и будет) пожимать плечами и обвинять тестировщиков. В конце концов, почему они выпустили продукт с таким количеством багов? С другой стороны, если тестировщики задерживают выпуск, то подвергаются пристальному вниманию и давлению из-за того, что являются такими фанатиками качества.

В конечном счете именно люди, контролирующие проект, лучше всего подготовлены к тому, чтобы нести ответственность за выпуск продук-

та. Однако в большинстве высокоэффективных проектов, которые мы видели, используется некий компромиссный вариант. Если вам дадут право контролировать выпуск продукта, мы рекомендуем сразу же настоять на том, чтобы эти полномочия были разделены с другими участниками команды.

Остерегайтесь выражения «не моя работа»

Тестирование — очень сложный процесс, тесно связанный с другими видами деятельности в проекте. Поэтому вам может показаться, что будет лучше сосредоточиться исключительно на тестировании. Некоторые тестировщики считают, что их цель заключается лишь в поиске отличий продукта от его описания в технических требованиях. Все, что выходит за рамки этой задачи, например проблемы с эксплуатацией, требованиями, качеством данных и поддержкой, — «не моя работа». Мы призываем вас взглянуть на ситуацию более широко. При прочих равных условиях ваша цель должна заключаться в том, чтобы в меру своих возможностей информировать команду о любых проблемах, которые могут негативно повлиять на ценность продукта. По этой причине в состав отличных команд тестировщиков входят самые разные люди, которые понимают, каким будет жизненный цикл продукта: как он будет разрабатываться, производиться, продаваться, использоваться, обслуживаться и обновляться.

Еще одно искушение сказать: «Это не моя работа» — возникает, когда вы оказываетесь в сложной ситуации тестирования. Ваши коллеги-программисты могут написать плохие спецификации. Или могут задержать доставку кода, и у вас не останется времени на нормальное тестирование. Они могут заявить, что проблема, которую вы нашли, — на самом деле плод вашего воображения. Такие обстоятельства могут побудить вас отказаться от тестирования. Вы можете сказать, что это не ваша работа — разбираться со спецификациями или проводить тестирование в спешке. В тяжелых случаях такой отказ может быть правильным решением, но сначала подумайте, насколько реалистичны ваши ожидания и есть ли другой способ получить желаемое. Если вы придерживаетесь принципа, что ваша работа — адаптировать и импровизировать (прилагая разумные усилия), то программисты, скорее всего, будут считать вас помощником, а не обузой. Это, в свою очередь, побудит *их* помогать *вам*.

Старайтесь не превратиться в команду улучшения процесса

Иногда вы устаете *искать* проблемы и начинаете задаваться вопросом, не лучше ли их *предотвратить*. Может быть, их было бы меньше, если бы программисты выполняли свою работу более тщательно. В этом есть смысл. С другой стороны, это все равно что учить жизни тех, кого вы любите. Если вы пробовали так делать, то знаете, что хороший совет не всегда воспринимается адекватно. Понимание здесь ни при чем, дело в отношении. Независимо от того, о чем идет речь, совершенствование процессов всегда связано с отношениями.

Даже если руководство поддержит вашу инициативу по улучшению качества, существует множество способов, которыми другие члены команды могут одновременно свести на нет ваши усилия и выставить вас некомпетентными. Да, вы можете участвовать в улучшении процессов и добиться успеха, если это *командная работа*. Но мы призываем вас не превращать команду тестировщиков в общество критиков процессов. Это контрпродуктивный путь.

Не ждите, что кто-то поймет, что такое тестирование или что вам нужно для качественного выполнения работы

Вы читаете эту книгу. Не ждите, что ее прочтет кто-то другой. Именно вы должны сообщить своим клиентам, что вам нужно для того, чтобы вы могли качественно выполнить свою работу. На вас влияет выбор, который делают руководство и программисты. Если они не уверены в своих планах или разрабатывают продукт, который трудно тестировать, то ваш тестовый проект может оказаться под угрозой срыва. Поговорите с этими людьми. Возможно, вы не получите всего, чего хотите, но дайте им шанс помочь вам.

Дело не в том, что этих людей не волнует тестирование или качество. Скорее всего, они просто не понимают, как их действия влияют на процесс тестирования. Важная часть роли тестировщика — объяснить клиентам суть процесса. Ваши объяснения подобны прививке от гриппа — она полезная и не очень болезненная, но ее воздействие со временем ослабевает, поэтому приходится делать ее снова и снова.

Глава 2

Мышление тестировщика

Тестировщиками становятся люди с разным опытом, и большинство людей сходятся в одном: тестировщики думают по-особенному. В чем это выражается? Некоторые говорят, что тестировщики мыслят «негативно», жалуются, любят все ломать и испытывают особое удовольствие, сообщая плохие новости. Это распространенная точка зрения. Мы же предлагаем другую. Тестировщики не жалуются — они предлагают доказательства. Тестировщики не любят ничего ломать — они любят развеивать иллюзию, что «все работает». Тестировщики не любят сообщать плохие новости — они любят освобождать своих клиентов от оков ложных убеждений. Мы считаем, что мыслить как тестировщик — значит заниматься эпистемологией. Тестирование — это прикладная эпистемология, а не брюзжание или нытье.

Эта глава представляет собой программу по превращению вашего разума в отточенный инструмент рассуждения. Используйте свои умственные способности во благо, а не во зло.

Тестирование — это прикладная эпистемология

Не пугайтесь! Мы не говорим о новой религии для кинозвезд. Эпистемология — это ветвь философии, которая *помогает нам тестировать лучше*.

Эпистемология изучает знание и то, как происходит его получение. Она исследует доказательства и аргументацию и закладывает основы научной практики. Эпистемологию изучают ученые, педагоги, философы и элитные тестировщики ПО. Изучающие ее студенты занимаются наукой, философией и психологией, чтобы узнать, как люди могут улучшить свое

мышление. Мы используем термин «эпистемология» в более широком смысле, чем его классическое определение, чтобы воспользоваться преимуществами последних работ в области критического мышления. Применительно к тестированию программного обеспечения эпистемология задает такие вопросы.

- Как вы узнаете, что программное обеспечение достаточно хорошее?
- Как вы узнаете, что оно недостаточно хорошее?
- Как вы узнаете, что провели достаточное количество тестов?

Сократ 2400 лет назад отстаивал идею критического рассмотрения веры и воплощал ее в жизнь. Поэтому мы считаем его ранним эпистемологом. Философы, ученые и психологи продолжают изучать эпистемологию и в наши дни. Это ваше наследие как тестировщика.

Изучение эпистемологии поможет вам тестировать лучше

Вот темы эпистемологии, которые непосредственно связаны с тестированием программного обеспечения:

- как собирать и оценивать доказательства;
- как делать обоснованные выводы;
- как использовать различные формы логики;
- что значит иметь обоснованное убеждение;
- различия между формальными и неформальными рассуждениями;
- распространенные ошибки в неформальных рассуждениях;
- смысл и неоднозначность в естественном языке;
- как принять правильное решение.

Многие люди, никогда не изучавшие данные темы, хорошо выполняли тестирование. Но если вы хотите быть *лучше*, чем просто хорошим тестировщиком, то разберитесь в этих темах. Изучение эпистемологии поможет вам разрабатывать эффективные стратегии тестирования, лучше распознавать ошибки в своей работе, понимать, что тестирование доказывает, а что — нет, и составлять убедительные отчеты о тестировании.

Вот три отличные книги, с которых стоит начать изучение мышления.

- *Tools of Critical Thinking: Metathoughts for Psychology* (Levy, 1997). Книга предназначена для психотерапевтов, но отлично подойдет и тестировщикам. В каждой главе описывается отдельная идея о том,

как мыслить лучше. Вам не обязательно читать все подряд, выберите любую понравившуюся главу.

- *Thinking and Deciding* (Baron, 1994). Книга, в которой в очень доступной форме рассказывается о мышлении. Послужит отличным введением в тему.
- *The Craft of Research* (Booth, Colomb, Williams, 1995). Предназначенная для студентов, эта небольшая книга посвящена критическому чтению и письму, в том числе построению убедительных аргументов.

Тестирование основано на когнитивной психологии

Если эпистемология рассказывает нам о том, как мы *должны* думать, то когнитивная психология — о том, как мы *думаем*. Вот некоторые из ее тем, связанных с тестированием:

- как работают наши разум и память;
- откуда берутся убеждения;
- как наши убеждения влияют на наше поведение;
- как предубеждения и упрощения влияют на принятие нами решений;
- как мы учимся и делимся тем, что знаем;
- как мы думаем о сложных вещах;
- как мы мыслим в стрессовой ситуации;
- как мы распознаем закономерности;
- как мы относим идеи и вещи к той или иной категории;
- как мы замечаем различия между вещами;
- как могут искажаться воспоминания о событиях;
- как мы реконструируем события, которые запомнили частично (например, невоспроизводимые баги).

Многие тестировщики, которые никогда не изучали эти темы, хорошо справлялись с тестированием. Если вы хотите тестировать не просто хорошо, а лучше, то изучение когнитивной психологии поможет вам понять, какие факторы влияют на вашу работу как тестировщика и на то, как люди ее оценивают.

Чтобы начать изучение этих тем, обратите внимание на книгу *Cognition in the Wild* (Hutchins, 1995), где автор изучает совместную работу команд военных моряков. Многое из этой книги применимо и к проектированию ПО и командам тестирования.

Полезная книга о психологии мышления — *Theory and Evidence: The Development of Scientific Reasoning* (Koslowski, 1996). Автор изучает, как люди используют теории причинно-следственных связей для рассуждения о системах. Это объясняет, почему процесс тестирования представляет собой нечто большее, чем просто наблюдение за внешним поведением программы и проверка его на соответствие простым ожиданиям.

Тестирование происходит в вашей голове

Разница между отличным и посредственным тестированием заключается в том, как вы мыслите: какой вариант создания тестов выбираете, способны ли объяснить наблюдаемые результаты и умеете ли убедительно рассказать об этом. В остальном же тестирование по большей части обычная офисная работа. Видя двух тестировщиков, работающих бок о бок, вы не всегда можете сказать, что один из них тестирует лучше другого. Видимая часть их работы выглядит одинаково, и из-за этого возникают два следствия:

- многие люди считают, что тестирование — это просто, поскольку они могут легко скопировать видимое поведение хорошего тестировщика и у них нет других стандартов продуктивного тестирования;
- если вы хотите быть хорошим тестировщиком, научитесь думать как тестировщик, а не выглядеть таковым.

Тестирование требует умозаключений, а не просто сравнения выходных данных с ожидаемыми результатами

Существует распространенное мнение, что тестировщики просто выполняют тестовые сценарии и сравнивают полученные результаты с ожидаемыми. При такой точке зрения тестирование превращается в простое сравнение, к тому же игнорируется тот факт, что какой-то умный человек должен *разработать* тесты и *определить* ожидания. И кто же это может быть? У этого разработчика тестов практически никогда нет доступа к авторитетному руководству, в котором описывается, что нужно тестировать, не говоря уже о том, чего следует ожидать. А имеющиеся руководства еще нужно правильно понять. В реальной жизни большинство тестов разрабатывается на основе умозаключений или опыта, который, по мнению тестировщика, имеет отношение к делу.

Более того, эти умозаключения меняются со временем. Думать как тестировщик означает владеть искусством исследовательских умозаключений.

Исследовательское умозаключение может показаться странной идеей. Оно означает, что одна идея ведет к другой, затем еще к одной, причем такими путями, которые вы не можете предсказать заранее. Отличное обсуждение исследовательского умозаключения можно найти в книге *Proofs and Refutations: The Logic of Mathematical Discovery*[1] (Lakatos, 1976). Самое замечательное в этой книге то, как Лакатос показывает, что процесс математических и научных рассуждений основан на исследовании, а не на сценариях. Даже математики рассуждают активно и пытливо, а не просто используют скучные формулы. Они думают как тестировщики!

Хорошие тестировщики думают технически, творчески, критически и практически

В практике тестирования используются все виды мышления. Но мы считаем, что о четырех основных категориях стоит сказать отдельно.

- *Техническое мышление* — способность моделировать технологию и понимать причины и следствия. Сюда же можно отнести знание соответствующих технических фактов, умение использовать инструменты и прогнозировать поведение систем.
- *Творческое мышление* — способность генерировать идеи и видеть возможности. Вы будете тестировать только теми способами, которые в состоянии представить. Вы сможете увидеть только те проблемы, которые, по вашему мнению, могут существовать.
- *Критическое мышление* — способность оценивать идеи и делать умозаключения. Сюда входит умение находить ошибки собственного мышления и ликвидировать их, соотносить наблюдения за продуктом с критериями качества, а также умение создавать адекватные аргументы в пользу того или иного убеждения или предлагаемого курса действий.
- *Практическое мышление* — способность воплощать идеи в жизнь. Подразумевает умение применять инструменты тестирования и согласовывать техники и объем работы по тестированию с границами проекта.

[1] *Лакатос И.* Доказательства и опровержения. Как доказываются теоремы.

В целом мышление тестировщика заставляет вас поверить в то, что все может быть не таким, как кажется. Как бы ни обстояли дела, они могли бы идти по-другому. Мы обнаружили, что, когда процесс тестирования оказывается неудачным, первопричиной, скорее всего, является узость взглядов. Другими словами, дело не в том, что мы провели 10 000 тестов, а должны были провести 10 001. Дело в том, что мы не смогли представить целую категорию тестов — тестирование, которое мы бы не провели, даже если бы имели вдвое больше времени и ресурсов.

Тестирование методом «черного ящика» — это не тестирование, основанное на незнании

Тестирование *методом «черного ящика»* означает, что знания о внутреннем устройстве продукта не играют существенной роли. Большинство тестировщиков используют именно этот метод. Чтобы хорошо провести такое тестирование, узнайте о пользователе, его ожиданиях и потребностях, технологии, о конфигурациях, которые будет использовать программное обеспечение, узнайте о другом ПО, с которым будет взаимодействовать ваше, о данных, которыми должно управлять ПО, о процессе разработки и т. д. Преимущество тестирования методом «черного ящика» заключается в том, что вы, вероятно, мыслите иначе, чем программист, и, следовательно, можете предвидеть риски, которые он упустил.

При тестировании методом «черного ящика» внимание акцентируется на изучении пользователя и среды ПО, что нравится далеко не всем. Мы даже слышали, что такое тестирование называют *тестированием, основанным на незнании*, поскольку тестировщик не взаимодействует с базовым кодом. Мы считаем, что такое мнение отражает фундаментальное непонимание роли команды тестировщиков. Мы не возражаем против того, чтобы тестировщик узнавал о том, как работает продукт. Чем больше информации о продукте вы получите, тем лучше сможете его протестировать. Сосредоточив основное внимание на исходном коде и способах его тестирования, вы пойдете по тому же пути, который, вероятно, уже пройден программистом, но при этом будете иметь меньше знаний о коде, чем он.

Тестировщик — больше, чем турист

Многое из того, что вы делаете с продуктом, по сути не является тестом, но может помочь вам изучить этот продукт. Вы можете осмотреть его,

увидеть, как он сделан и как работает. Это бесценно, но это не совсем тестирование. Разница между тестировщиком и туристом заключается в том, что тестировщик старается оценить продукт, а не просто наблюдать за ним. Предсказывать заранее, как должно вести себя программное обеспечение, нет необходимости. Тем не менее действие, которому подвергается продукт, не становится тестовым до тех пор, пока вы не примените некий принцип или процесс, позволяющий выявить какую-либо проблему, если она существует.

Любой тест — это попытка ответить на какой-либо вопрос

Все тесты — это эксперименты, которые проводятся для того, чтобы ответить на вопрос: как соотносится то, что представляет собой продукт, и то, каким он должен быть. Иногда вы не совсем понимаете, какие вопросы задаете. Это может быть нормально, если вы ищете только *очевидные* проблемы. Однако во многих случаях баги не бросаются в глаза и не просят сообщить о них, вывешивая мигающую неоновую надпись «Внеси меня в отчет». Продукт может вести себя неправильно в ситуациях, которые пользователь сочтет очевидными, даже если вы (тестировщик) этого не замечаете. При любом тестировании спросите себя, какие вопросы должны лежать в основе вашей стратегии оценки. В противном случае вы будете больше туристом, чем тестировщиком.

Все тестирование основано на моделях

При разработке тестов вы можете держать в голове мысленную картинку. Возможно, вы работаете со списком функций или какой-то диаграммой. У вас есть некое представление о том, кто такие пользователи и что их волнует. Все это — модели. Несмотря ни на что, ваши тесты будут основываться в первую очередь на ваших моделях продукта, а не на нем самом. Некорректная модель приводит к ошибочным тестам. Изучить новый способ моделирования продукта — все равно что научиться видеть его по-новому.

Изучайте моделирование. Овладевая этим искусством, вы будете тестировать все лучше и лучше. Вам могут помочь учебники и курсы по анализу требований и архитектуре программного обеспечения. Замечательный способ приобрести навыки во всех видах моделирования — изучить системное мышление. См. книгу *An Introduction to General Systems Thinking: Silver Anniversary Edition* (Weinberg, 2001).

Интуиция хороша в начале и плоха в конце

У вас может возникнуть соблазн использовать те или иные тестовые данные или оценивать тот или иной результат, основываясь на интуиции — чувстве, что вы *знаете*, даже если не можете изложить причины, обосновывающие это знание. Мы считаем, что это чувство полезно, но скорее в качестве отправной точки, чем точки опоры.

Учитывая, что интуиция часто бывает предвзятой, настоящие проблемы возникают, когда вы пытаетесь заставить других людей, например программистов и руководителей, серьезно относиться к вашим баг-репортам и оценкам качества. Если ваши выводы не основываются на чувстве, которое есть у всех участников процесса, то вашу работу могут отклонить.

Поэтому мы рекомендуем использовать интуицию как руководство к действию, но не как *оправдание*. Когда у вас возникает соблазн подумать: «Это баг, поскольку это очевидно», попробуйте переформулировать вашу мысль так: «Это баг, поскольку я наблюдаю, что продукт ведет себя таким образом, что нарушает требования X, Y и Z, которые ценны для моих клиентов».

Чтобы тестировать, вы должны исследовать

Чтобы хорошо протестировать что-либо, нужно с этим поработать. Вы должны *вникнуть* в предмет тестирования. Это исследовательский процесс, даже если у вас есть идеальное описание продукта.

Пока вы не изучите эту спецификацию либо мысленно, либо работая с самим продуктом, придуманные вами тесты будут поверхностными. Даже после того, как вы изучите продукт настолько хорошо, чтобы глубоко его понять, остается вопрос поиска проблем. Любое тестирование — это выборка, а она никогда не может быть полной. Поэтому исследовательское мышление играет важную роль на протяжении всего тестового проекта, поскольку вы стремитесь максимизировать ценность тестирования.

Под исследованием мы понимаем целенаправленное блуждание: перемещение по пространству с определенной целью, но без заданного маршрута. Исследование предполагает постоянное изучение и экспериментирование. При этом возможны многократные возвраты, повторения и другие процессы, которые для неподготовленного человека

выглядят как напрасная трата ресурсов. Возможно, по этой причине важность исследования в сфере тестирования, да и в программной инженерии, часто преуменьшается и даже высмеивается писателями и консультантами в нашей области.

В книге мы не ставили задачу доказать принципиальную важность исследований. Один из способов наглядно убедиться в этой важности — понаблюдать за тем, как вы пытаетесь собрать пазл, не глядя на картинку на коробке, или играете в угадайку или «Мастермайнд». Обратите внимание, насколько сложнее добиться успеха в таких играх и насколько менее полезным будет результат, если действовать строго по заранее заданному сценарию.

Чтобы узнать, как рациональное исследование используется в социологии — области, во многом схожей с тестированием, — обратитесь к книгам *The Discovery of Grounded Theory: Strategies for Qualitative Research* (Glaser, Strauss, 1999) и *Basics of Qualitative Research, 2nd Edition*[1] (Strauss, Anselm, Corbin, 1998). Если вам нравится статистика, то попробуйте прочесть *Exploratory Data Analysis*[2] (Tukey, 1977).

Исследование требует многих размышлений

Исследование — это детективная работа. Это бесконечный поиск. Думайте об исследовании как о движении в пространстве. В этом случае задействуется прямое, обратное и нестандартное мышление.

- *Прямое мышление.* Двигайтесь от того, что знаете, вперед к тому, чего не знаете; от того, что видите, к тому, что еще не видели. Ищите последствия и побочные эффекты. Пример: «Я вижу пункт меню “Печать”. Я щелкну на нем и посмотрю, что произойдет».
- *Обратное мышление.* Двигайтесь от того, что вы подозреваете или представляете, назад к тому, что знаете, пытаясь подтвердить или опровергнуть свои догадки. Пример: «Интересно, есть ли способ распечатать этот документ? Я посмотрю, есть ли в меню пункт “Печать”» (Solow, 1990).
- *Нестандартное мышление.* Позвольте себе отвлекаться на идеи, которые приходят вам в голову, исследуйте их, а затем возвращайтесь к основной теме (de Bono, 1970). Пример: «Интересный график. Распечатаю-ка я его и посмотрю, что получится».

---

[1] *Страусс А., Корбин Дж.* Основы качественного исследования: обоснованная теория. Процедуры и техники.

[2] *Тьюки Дж.* Анализ результатов наблюдений. Разведочный анализ.

Процесс исследования работает, даже если у вас нет продукта для тестирования. Вы можете изучить документацию или поговорить с программистом, используя те же самые мыслительные процессы. Вы добиваетесь прогресса, создавая все более полные и совершенные мысленные модели продукта. Затем они позволяют разрабатывать эффективные тесты.

Используйте логику абдуктивного умозаключения, чтобы найти гипотезы

Абдуктивное умозаключение, также известное как гипотетическая индукция, — замысловатый термин для обозначения очень важной формы рассуждений, которую тестировщики используют каждый день: *рассуждение для поиска наилучшего объяснения*. Это происходит следующим образом.

1. Вы собираете некие данные и хотите в них разобраться.
2. Вы строите различные предположения, которые могут объяснить эти данные.
3. Вы ищете дополнительные данные, которые помогут вам подтвердить или опровергнуть каждое из предположений.
4. Вы выбираете из всех вариантов *самое понятное предположение*, учитывающее все важные данные, или продолжаете поиск, если доказательств, подтверждающих какой-либо вывод, недостаточно.

Абдуктивное мышление — базовый метод, использующийся в науке и тестировании. Врачи с его помощью диагностируют болезни. Тестировщики используют его, когда делают предположения о том, чем является продукт и как он должен или не должен работать. Если вы хотите строить абдуктивные умозаключения более продуктивно, то:

- соберите больше данных;
- соберите больше важных данных;
- соберите больше достоверных данных;
- изучите причины и следствия, применимые к этим данным;
- выдвиньте больше хороших предположений, которые объяснят данные;
- соберите больше данных, которые опровергнут каждое предположение;
- соберите больше данных, которые позволят различать предположения;
- не останавливайтесь на предположении, если оно не учитывает все важные данные и не является явно более обоснованным, чем другие.

Абдукция — систематический метод для поиска хороших предположений. Хотя процесс абдуктивного умозаключения не гарантирует абсолютной достоверности, это лучший метод, которым мы располагаем в большинстве ситуаций.

Используйте логику гипотезы и опровержения для оценки продукта

Философ Карл Поппер ввел метод гипотезы и опровержения в начале XX века, когда работал над проблемой различий религии и науки (Popper, 1989). Метод основан на предположении, что ученый никогда не может быть абсолютно уверен в каком-либо конкретном факте или теории о природе. Все существующее — гипотезы. Некоторые из них, такие как существование гравитации, очень сильны. Гипотезой, а не абсолютно достоверным фактом их делает то, что можно представить себе новую информацию, которая, существуй она, заставила бы нас опровергнуть гипотезу. Поппер заметил, что мы не можем доказать, что гипотеза верна, однако *можем* доказать, что она ложна. Поэтому он предположил, что для подтверждения гипотезы нужно стремиться опровергнуть ее и что мы не можем этого сделать.

Такой метод создания гипотез и попыток их опровергнуть применим в тестировании в трех важных аспектах.

- Тестирование, имеющее целью показать, что продукт *не работает*, гораздо эффективнее тестирования, которое проводится с противоположной целью. Когда вы хотите узнать, хорошо ли работает продукт, найдите способы опровергнуть предположение о том, что он работает хорошо, — и ваше тестирование, вероятно, будет более эффективным.
- Хорошо сформированное представление о программном обеспечении (как оно себя ведет, насколько оно хорошее и т. д.) должно быть *опровергаемым*. Это означает, что мы должны уметь представить себе новую информацию, которая противоречила бы нашим представлениям. В противном случае наше представление не более чем вера. Она хороша в личной жизни, но для тестирования губительна.
- Остерегайтесь тестов, целью которых является проверка или сертификация продукта способами, которые выходят за рамки проведенных вами конкретных тестов. Никакое количество тестов не может дать *уверенности* в качестве продукта.

Требование — это качество или условие, имеющее значение для тех, кто принимает решение

Вы можете выбрать одно из многих определений понятия «требование». Определение в заголовке хорошо подходит для тестировщиков. В процессе тестирования вы должны понимать, чье мнение о качестве продукта имеет значение (не все из мнений имеют одинаковое значение). Затем узнайте у этих людей, что они хотят видеть в продукте и чего не хотят. Такой взгляд на требования не делает различий между разработкой программного обеспечения «по требованиям» (набору утверждений, опубликованных в «документе требований» и одобренных людьми, имеющими на это право) и любым другим видам спецификаций. Для целей тестирования любое качество или условие, которое должен демонстрировать или соблюдать продукт, является требованием.

У разных клиентов разные требования к продукту, они не всегда знают, чего хотят, и с течением времени их пожелания меняются. Это делает нашу работу более интересной. Добро пожаловать в мир тестирования!

Вы выясняете требования, используя обсуждения, умозаключения и справочные документы

Если вы ждете, что получите требования на куске пергамента, скрепленного печатью абсолютной истины, то найдите другую работу. В *лучших случаях*, с которыми мы сталкивались, документация по требованиям (к которой относятся всевозможные спецификации продукта, сценарии использования, документ-концепция и т. д.) оказалась неполной и неоднозначной, даже если была информативной и полезной. В худших случаях документация была неполной, неоднозначной, *не*информативной и *не*полезной.

Тестировщик, который использует проектную документацию (спецификации продукта) в качестве *единственного* источника требований, вредит процессу тестирования. В каждой команде тестировщиков, которой мы руководили, подобное поведение было бы чревато увольнением.

Информация о требованиях поступает к нам через три основных источника:

- *обсуждения* — пообщайтесь с человеком, чье мнение о качестве имеет значение, и узнайте, что для него важно;

- *умозаключения* — определите, какие требования важны, экстраполируя их на другие известные вам сведения о проекте и продукте;
- *справочные документы* — ознакомьтесь с явными и неявными спецификациями и основывайте свое тестирование на них.

Во многих проектах большинство требований, которые используют хорошие тестировщики, формируются либо на основе умозаключений, либо на основе справочных документов с неявными спецификациями. Это ваша задача — найти информацию, необходимую для тестирования.

Отличная книга об этом: *Exploring Requirements: Quality Before Design* (Gause, Weinberg, 1989).

Используйте как явные, так и неявные спецификации

Не все справочные документы, содержащие важную информацию, на которой вы можете основывать свои тесты, могут быть представлены вам явно:

- *явная спецификация* — полезный источник информации о требованиях, *признанный* клиентами *в качестве авторитетного* («Да, это спецификация. Это описание продукта»);
- *неявная спецификация* — полезный источник информации о требованиях, который *не признается* клиентами в качестве авторитетного («Это не спецификация, но имеет смысл»).

Авторитетность неявной спецификации базируется на убедительности и достоверности содержащихся в ней сведений, а не на похвале клиентов. В большинстве случаев только часть неявной спецификации имеет отношение к рассматриваемому продукту. Неявные спецификации могут принимать разные формы:

- конкурирующие продукты;
- сопутствующие продукты;
- более старые версии продукта;
- обсуждения по электронной почте в рамках проекта;
- комментарии клиентов;
- журнальные статьи (например, обзоры старых версий продукта);
- учебники по смежным предметам (бухгалтерская книга может относиться к бухгалтерской программе);
- руководства по стилю графического пользовательского интерфейса;

- требования к совместимости операционных систем;
- ваш личный опыт, основанный на фактах.

Если продукт нарушает явную спецификацию, то перед вами встает относительно простая задача сообщить: «Он нарушает спецификацию, поэтому продукт, скорее всего, неправильный». Когда же нарушается неявная спецификация, приходится приводить более веские аргументы: «В Microsoft Office клавиша F4 связана с командой *repeat*. Если мы не сделаем то же самое, то можем запутать наших пользователей, которые используют Office в повседневной работе». Хотя никто не скажет, что Microsoft Office *является* спецификацией для *вашего* продукта, ваши клиенты могут согласиться с тем, что приведение пользовательского интерфейса в соответствие с Office улучшит удобство использования. Если это так, то Office является неявной спецификацией для вашего продукта.

Некоторые тестировщики задаются вопросом, почему разработчики просто не вставят все полезное в явные спецификации, чтобы им не приходилось отличать спецификации от неявных ресурсов. Ответ простой: хотя это было бы удобно для тестировщиков, это дорого и ненужно. Наши клиенты поручают нам использование любых ссылок, позволяющих быстро обнаруживать важные проблемы.

«Это работает» на самом деле означает, что «это в какой-то степени соответствует каким-то требованиям»

Каждый раз, когда вы слышите, как кто-то говорит: «Я попробовал, и это сработало», «Я уверен, что это работает» или «Теперь это работает лучше», мы рекомендуем вам перевести «это работает» в «похоже, это в какой-то степени соответствует каким-то требованиям». Вы сразу можете задуматься вот над какими вопросами.

- Что «это»? О какой части продукта мы говорим?
- Как это проявляется? Что именно наблюдалось?
- Какие требования были проверены? Функциональность? Производительность?
- В какой степени было выполнено требование для прохождения теста? Это сработало нормально или как нельзя лучше?
- Когда это работает? Какой диапазон обстоятельств был охвачен тестом? До какой степени эти обстоятельства можно безопасно обобщать?

Если не хотите, то можете не задавать эти вопросы вслух. Дело в том, что фраза «это работает» без дальнейших уточнений двусмысленна. Ваше мнение по поводу «это работает» может не совпадать с чьим-то другим определением.

В конце концов, у вас есть лишь представление о продукте

Что бы вы ни знали о качестве продукта — это предположения. Независимо от степени их обоснованности, вы не можете быть уверены в своей правоте. Поэтому каждый раз, когда сообщаете об уровне качества продукта, вы должны дополнить отчет информацией о том, как проводили тестирование, и об известных ограничениях вашего процесса тестирования.

Не путайте тесты и тестирование

Что означает создание тестов? Это может означать, что тестировщик провел сеанс исследовательского тестирования, в результате которого были созданы эфемерные тесты без документации и тестового продукта. Это может означать, что тестировщик создал набор исполняемых тестовых программ или явных тестовых процедур. Это может относиться к высокоуровневой матрице тестирования, схеме проведения тестирования или набору тестовых данных.

Концепция теста, который является самодостаточным, осязаемым и отличающимся от других, удобна (мы будем использовать ее на протяжении всей книги, поскольку это стандартный жаргон нашей сферы), но она же и ограничивает. Важно именно тестирование, а не то, как вы разбиваете его на модули, которые называете тестами. Тестирование — это все, что включает в себя как минимум четыре вида деятельности.

- *Конфигурируйте.* Подготовьте продукт к тестам. Приведите его в правильное начальное состояние. В противном случае нестандартные переменные могут испортить результаты тестов.
- *Выполните.* Введите тестовые данные для продукта. Подавайте команды. Каким-то образом взаимодействуйте с продуктом. Иначе вы будете лишь сидеть и наблюдать, а не тестировать.
- *Наблюдайте.* Собирайте информацию о поведении продукта, выходных данных, состоянии системы в целом, взаимодействиях с другими продуктами и т. д. Вы не можете наблюдать за всем, но все, что вы наблюдаете, может помочь обнаружить баги.

- *Оценивайте.* Применяйте правила, рассуждения, механизмы, которые позволят обнаружить баги в наблюдаемых данных. В противном случае вы либо не сообщите о проблемах, либо просто передадите данные своим клиентам, которые должны будут провести оценку самостоятельно.

Создание тестов может принимать различные формы. Не зацикливайтесь на форме, просто убедитесь в том, что указанные четыре действия выполняются. Сосредоточьтесь на том, кто их делает и насколько хорошо тесты соответствуют намеченной стратегии и цели тестирования.

При тестировании сложного продукта погружайтесь и откладывайте

Иногда тестирование может оказаться невероятно сложным. Вы можете почувствовать, что ничего не понимаете. Поэтому тестируйте сложный набор функций небольшими сериями. Человеческий разум обладает удивительной способностью справляться с непростыми задачами, но не ждите, что сможете разобраться в сложном продукте сразу. Попробуйте поработать над ним 30 минут или час. Затем прервитесь и займитесь чем-нибудь другим. Так выглядит метод «погрузиться и отложить». Не беспокойтесь о том, что в течение этого короткого перерыва вы будете непродуктивны; если чувствуете себя слишком растерянными, то заканчивайте серию раньше.

Преимущество такого метода заключается в том, что он не требует абсолютно никакого плана: вам нужно только выбрать часть продукта и работать с ней. Спустя несколько циклов «погружения» и «откладывания» вы будете иметь общее представление о продукте. Вскоре в голове появятся более организованные и конкретные стратегии тестирования и изучения. Это работает как по волшебству. В конце концов вы будете знать достаточно, чтобы разработать комплексный тест-план, если считаете, что это послужит вашей цели.

Используйте эвристику, чтобы быстро генерировать идеи тестирования

Эвристика — это совокупность исследовательских методов, помогающих сделать обоснованное предположение. Слово происходит из греческого языка и означает «служащий для обнаружения». Эвристика

не гарантирует получения правильного или наилучшего ответа и тем не менее является полезной. Из книг, посвященных использованию эвристики, особенно рекомендуем *How to Solve It*[1] (Polya, 1957).

Поскольку количество возможных тестовых случаев бесконечно, мы вынуждены делать предположения о том, какая небольшая их группа будет эффективной в условиях наших временны́х и бюджетных ограничений. Опытные тестировщики собирают эвристические приемы тестирования, улучшающие качество их предположений, и обмениваются ими. Хороший набор приемов позволяет создавать тесты очень быстро. Приведем несколько примеров таких приемов.

- *Тестируйте граничные значения.* С большей вероятностью это позволит выявить неясности в спецификации.
- *Проверяйте каждое сообщение об ошибке.* Код обработки ошибок, как правило, слабее кода основной функциональности.
- *Тестируйте конфигурации, которые отличаются от созданных программистом.* Работу своей конфигурации программист уже проверил.
- *Выполняйте тесты со сложной настройкой.* При прочих равных условиях с большей вероятностью будут выполнены простые в настройке тесты.
- *Избегайте избыточных тестов.* Если один тест повторяет другой, то что вам это даст?

Но помните: эвристика лишь дает вам пищу для размышлений. Не стоит слепо следовать эвристическим приемам, которые вы не понимаете. Ни к чему хорошему это не приведет. Отбирая приемы, постарайтесь понять причины, лежащие в основе каждого из них, и условия, при которых он с большей или меньшей вероятностью будет работать.

Избежать предубеждений невозможно, но можно управлять ими

Вы предвзяты. Это заставляет вас с большей вероятностью выбирать одни, а не другие тесты. Если есть поле для редактирования, то, скорее всего, вы введете в него что-то вроде 1111111111, а не 3287504619, поскольку ввести строку из повторяющихся символов проще, чем цифры от 0 до 9 в случайном порядке. Возможно, эта вероятность

[1] *Пойа Д.* Как решать задачу.

невелика, но она существует. Более зловещим является тот факт, что большинство тестировщиков склонны тестировать самые заметные функции, независимо от того, важны ли они. Кроме того, многие тестировщики тяготеют к очень простым и искаженным входным данным, а не к реалистичным данным средней сложности, и считают, что то же предпочитают и пользователи.

Приведем несколько популярных примеров предубеждений.

- *Ассимиляционные предубеждения.* Более вероятно, что я буду интерпретировать результаты будущих тестов так, чтобы они подтверждали мое мнение о продукте.
- *Подтверждение.* Я с большей вероятностью обращу внимание на результаты тестирования, которые действительно подтверждают мое мнение о продукте.
- *Доступность.* Если я могу легко представить сценарий, в котором пользователь ведет себя определенным образом, то буду считать, что такое поведение наиболее вероятно.
- *Первенство.* Я буду больше доверять мнению, составленному во время первого наблюдения.
- *Новизна.* Я буду больше доверять мнению, составленному во время недавнего наблюдения.
- *Эффект обрамления.* Когнитивное искажение, при котором форма подачи информации влияет на ее восприятие человеком. На мою реакцию на баг-репорт сильно влияет то, как он сформулирован, независимо от того, что он означает.
- *Известность.* Я буду уделять больше внимания мнению пользователей, с которыми я знаком.
- *Репрезентативность.* Я ожидаю, что маленькие проблемы вызваны незначительными причинами, а большие проблемы порождены серьезными причинами.

Избежать этих предубеждений невозможно. Они глубоко укоренились в нашем мозге. Но ими можно управлять. Например, просто изучая, в чем вы необъективны, и практикуясь в осознании этого, вы сможете избавиться от предубеждений. Защититься от излишней предвзятости можно также с помощью разнообразия. Если несколько тестировщиков решают проблемы с тестами сообща, то это может свести к минимуму влияние предубеждений одного тестировщика.

По определению, эвристика — тоже предубеждение. Мы используем эвристические приемы, поскольку надеемся, что они будут полезны.

Вас труднее обмануть, если вы знаете, что вас можно обмануть

Мошенники говорят, что легче всего обмануть того, кто убежден, что обмануть его невозможно. Вы можете применить этот принцип в своей работе тестировщика. Убедите себя в том, что вас *легко* обмануть. Это несложно, просто внимательно следите за своими ошибками во время тестирования. Замечайте моменты, когда другой тестировщик обнаружил проблему, которую вы могли найти, но не нашли.

Если вы знаете, что вас легко обмануть, то станóвитесь более бдительными. Вы тщательнее прорабатываете свою тестовую стратегию. Это один из самых быстрых способов повышения квалификации для начинающего тестировщика. Проблема начинающих тестировщиков в том, что для них этот принцип — просто символ веры («Мне сказали, что я должен думать, что меня можно обмануть. Ну ладно...»). А чувства и рефлексы опытных тестировщиков оттачивались в полевых условиях, при устранении реальных ошибок («Я помню великий сбой в 94-м. Мы не могли представить, что вирус может попасть на наш главный диск. В тот день я стал настоящим тестировщиком»).

Если вы пропустили баг — проверьте, вышло ли это случайно или это естественный результат вашей стратегии тестирования

Если вы подбрасываете монету и загадываете орла, но выпадает решка, значит ли это, что вы приняли неверное решение? Если это не монетка фокусника, то вероятность того, что выпадет орел или решка, — 50 на 50. В том, что выпала решка, нет ничего удивительного, просто вам не повезло. Ваша стратегия в порядке.

Такая же проблема возникает, когда вы не находите баг в процессе тестирования, из-за чего у ваших клиентов случаются неприятности. Не корите себя за это, пока не разберетесь, что произошло со стратегией тестирования. Вы пропустили баг, поскольку неукоснительно следовали хорошей стратегии тестирования и просто не нашли эту конкретную ошибку? Если да, то продолжайте в том же духе. Такое случается. Но если вы пропустили баг, поскольку в вашей стратегии тестирования внимание уделяется другим проблемам, то используйте эту возможность, чтобы улучшить ее.

Замешательство — инструмент тестирования

Если вы в замешательстве, то знайте: это чувство может говорить вам о чем-то важном.

- *Замешательство вызывают спецификации?* Неясности в них часто вызваны тем, что таким образом могут скрываться важные разногласия между влиятельными стейкхолдерами.
- *Замешательство вызывает продукт?* Он может не работать.
- *Замешательство вызывает пользовательская документация?* Эта часть продукта может быть очень сложной. Специфических случаев и несоответствий между ними слишком много, чтобы их можно было описать.
- *Трудно понять основную задачу?* Некоторые системы, которые мы пытаемся автоматизировать, по своей сути сложны или связаны с непростыми техническими проблемами. Программисты тоже считают их сложными, и это приводит к упущениям, непониманию, чрезмерному упрощению.

Чем больше вы узнаёте о продукте, технологии и тестировании в целом, тем более мощным компасом служит вам замешательство, показывая, в каких местах кроются важные проблемы.

В ходе тестирования, если вы больше ничего не знаете о продукте, вы по крайней мере понимаете, что находитесь в замешательстве. В такой ситуации появление этого чувства может стать вашим лучшим результатом, помогая поднять вопросы, которые, возможно, никто другой не осмелился задать.

Ошибка видна свежим взглядом

Осмысление чего-либо — сложный интеллектуальный процесс, в ходе которого новая информация добавляется к уже известной, а та, в свою очередь, изменяется под влиянием новых данных. После того как вы разобрались с продуктом или функцией, у вас появляется представление о них и ваш мозг работает уже не так напряженно. Это может стать проблемой для тестировщиков. Когда вы хорошо знаете продукт, вы строите больше предположений о нем и реже их проверяете.

Эта ситуация чревата по крайней мере тремя последствиями для тестирования.

- Когда вы впервые сталкиваетесь с продуктом или функцией, обратите особое внимание на то, что вас смущает или раздражает. Это может подсказать вам, как отреагирует пользователь.
- Если у вас в команде новички — работайте вместе с ними. Наблюдайте за их реакцией на продукт и на то, как они изучают его.
- Остерегайтесь привычных способов тестирования. Даже если вы не следуете жестким сценариям тестирования, вы можете настолько глубоко изучить определенную функцию, что будете тестировать ее все более ограниченными способами. Чередуйте техники везде, где это возможно, или поменяйтесь обязанностями с другим тестировщиком.

Избегайте выполнения процедур, в которых не уверены

Остерегайтесь процедур других людей. Обычно тест-кейсы и процедуры сформулированы так, что в них ничего не говорится о целях, лежащих в основе разработки теста. Повышается вероятность того, что вы будете выполнять тесты, не понимая, как их настраивать и на что обращать внимание. Другими словами, на самом деле вы не будете их выполнять. Как правило, процедуры тестирования плохо написаны и плохо разработаны, поскольку мало кто из хороших тестировщиков умеет программировать людей, как компьютеры. Если вы собираетесь следовать процедурам тестирования, то выбирайте те, которые разработали сами, те, которые принадлежат вам, или те, которые вы досконально изучили.

Чтобы достичь наилучших результатов, *вы* должны контролировать тестирование, а не ваша документация.

Если вы убеждены, что процедуры полезны, то по крайней мере изучите, как они работают. См. книги *Things that Make Us Smart: Defending Human Attributes in the Age of the Machine* (Norman, 1993) и *The Social Life of Information* (Brown, Duguid, 2000).

Если вы создаете процедуры тестирования, то опасайтесь «1287»

Один из нас, Бах, однажды был свидетелем того, как тестировщик писал процедуру тестирования, которая содержала строку «Введите 1287 символов в поле». Откуда взялось 1287? Тестировщик объяснил:

идея заключалась в том, чтобы ввести очень большой набор символов в маленькое поле ввода. И поскольку он слышал, что процедуры тестирования должны быть конкретными, то вернулся и тщательно подсчитал количество введенных им символов — их оказалось 1287. Он вписал его в процедуру — и теперь это произвольное число закреплено в ней навсегда.

Излишняя конкретизация бесполезна. При написании процедуры тестирования избегайте любых уточнений, не относящихся к концепции теста. Добавляйте любую информацию и конкретику, которые нужны для того, чтобы сформулировать и объяснить тест, но позвольте будущему тестировщику проявить творческий подход и здравомыслие. Дайте ему возможность внести изменения, которые обновят процедуру тестирования и сделают ее более эффективной.

Один из важных результатов процесса тестирования — появление более компетентного и умного тестировщика

Мы часто слышим аргументы против любых форм тестирования, в результате которого документация сведена к минимуму или ее вовсе нет, как будто единственная ценность тестирования состоит в описании тестов. При этом игнорируется очень важный участник тестирования — собственно тестировщик.

Хорошие тестировщики всегда учатся. По мере развития проекта они изучают продукт и постепенно улучшают навыки и понимание всех деталей, имеющих значение в проекте. Опытный тестировщик, который прошел один или два цикла релиза, может проводить тестирование гораздо эффективнее (часто без всяких инструкций), чем неопытный, которому вручают набор письменных инструкций о том, как тестировать продукт.

Некоторые консультанты и писатели в нашей области, кажется, верят в то, что неэффективный тестировщик может стать эффективным, если будет следовать процедурам тестирования. На наш взгляд, это плохая практика. Она отображает основополагающее заблуждение о тестировании и о людях, которые умеют хорошо тестировать.

Оценивая процесс тестирования, в первую очередь обратите внимание на качество работы тестировщиков проекта. Посмотрите, как они мыслят и как это влияет на их действия. Только после этого вы сможете оценить результаты их работы.

Вы не сможете освоить тестирование, если не изобретете его заново

Не изобретайте колесо. Хотя подождите. *Разве не колесо изобретают снова и снова?* Это же *хорошо*? В конце концов, сейчас мы ездим на пневматических шинах, а не на деревянных колесах. Существуют тысячи, если не миллионы вариаций на тему колеса. Может быть, это урок? Нам кажется, есть как минимум две причины изобретать что-то заново: адаптировать это нечто к новой ситуации и изучить, как оно работает. Мастерство требует и того и другого.

У нас есть коллеги, которые советуют студентам, изучающим тестирование, *избегать* изобретения тестов или идей тестирования. Мы с этим не согласны. Поступать так — все равно что изучать науку, избегая экспериментов. Учиться у других — нормально. Мы считаем, что очень важно учиться таким образом, — если не верите, посмотрите на название этой книги. Но если вы будете учиться *только так* — вы никогда не станете мастером в области тестирования. Вы будете подмастерьем, не более. Следование инструкциям не поможет вам овладеть навыками, так же как движение по шоссе между штатами не приведет вас на Марс. Мы призываем вас учиться тестированию так, как это делают великие механики и великие программисты: разбирайте вещи до винтиков, задумывайтесь над тем, как они работают, и собирайте по-новому. Не ограничивайте себя ролью хранителя чужого опыта, будьте автором своего.

На ранних этапах процесса обучения ваши заново изобретенные тесты, идеи, методики или документы будут не очень хорошими. Это нормально. Просто не выключайте мозг, наблюдайте за другими тестировщиками, учитесь и постоянно оценивайте результаты ваших идей. Если вы хотите добиться хороших результатов, то вам нужно практиковаться.

Мы занимаемся тестированием уже много лет и все еще изобретаем, все еще перерабатываем старые идеи. Каждый уважаемый нами коллега работает над достижением мастерства именно так.

Глава 3

Техники тест-дизайна

Чем занимается тестировщик? В первых двух главах наши ответы были мудрыми и познавательными, но, как мы надеемся, довольно абстрактными. Настало время перейти к конкретике. Откуда берутся тесты? Как они выглядят? Эта глава посвящена *техникам тестирования*, но мы не будем подробно описывать каждую. Эта информация приведена в основных учебниках по тестированию. Мы рекомендуем книги Кейнера, Фолка и Нгуена (Kaner, Falk, Nguyen, 1993), Йоргенсена (Jorgensen, 1995), Бейзера (Beizer, 1990), Марика (Marick, 1995) и Колларда (Collard, 2001). Полезные идеи также содержатся в материалах Уиттакера и Йоргенсена (Whittaker, Jorgensen, 1999, 2000) и Уиттакера (Whittaker, 2002).

Эта глава имеет два отличительных признака.

- Во-первых, важнейшая идея главы — структурная система классификации, по которой организован остальной материал. Мы изложили ее в первом уроке. В последующих пяти перечисляется несколько методик, но основная цель этих перечислений — поддержать систему классификации. Мы приводим эти сведения для того, чтобы вам было легче представить, как применять систему в своей работе.

  Эта система классификации синтезирует подходы, которые мы использовали и преподавали по отдельности. С помощью этой структуры вы можете принимать решения о том, какие техники доступны и подходят для решения конкретной задачи, а также генерировать идеи по комбинированию техник, предназначенных для эффективного решения поставленной задачи.

  Перечни техник иногда содержат подробности, выходящие за рамки краткого описания, но мы считаем такие пояснения необязательными. Уровень детализации намеренно неравномерный. Мы пред-

полагаем, что подробную информацию о большинстве техник вы получите в других книгах и курсах.

- Во-вторых, несмотря на то что эта глава не является практической, мы не могли заставить себя написать главу о техниках тестирования, не описав хотя бы несколько из них настолько подробно, чтобы вы могли их реально использовать. Поэтому в разделе «Дополнение к техникам тест-дизайна» представлены пять техник, которые мы считаем полезными и которые хорошо зарекомендовали себя у наших студентов на профессиональных семинарах и университетских курсах по тестированию ПО.

При выборе техники тестирования нужно думать о тестировщиках, покрытии, потенциальных проблемах, действиях и оценке

Основная цель этой главы — представить систему классификации техник тестирования. Мы называем ее *пятимерной системой тестирования*. Любой вид *тестирования* описывается с учетом пяти критериев.

- *Тестировщики. Кто* занимается тестированием. Например, пользовательское тестирование ориентировано на тестирование представителями целевой аудитории — теми, кто обычно использует продукт.
- *Покрытие. Что* тестируется. Например, при функциональном тестировании проверяется каждая функция.
- *Потенциальные проблемы. Почему* вы проводите тестирование (какие риски проверяете). Например, тестирование на ошибки при обработке экстремальных значений.
- *Действия. Как* вы тестируете. Например, исследовательское тестирование.
- *Оценка. Как определить, что тест прошел или упал.* Например, сравнение с известным правильным результатом.

В этой главе мы также подробно описали несколько техник и представили свои соображения по поводу использования некоторых других, но наша главная цель — объяснить систему классификации.

Все тесты содержат все пять критериев. Техника тестирования ориентирована на один из пяти критериев, при этом четыре других остаются на ваше усмотрение. Чтобы достичь желаемого результата, вы можете комбинировать техники, каждая из которых посвящена рабо-

те с каким-то одним критерием. Вы можете назвать результат такого объединения новой техникой тестирования (некоторые так и делают). Но мы считаем, что важнее развивать творческое мышление, чем бессистемно добавлять новые названия в и без того постоянно расширяющийся перечень техник. Наша схема классификации поможет создавать такие комбинации осознанно и продуманно.

Задачи тестирования часто концентрируются на одном критерии, но вы работаете со всеми пятью. Например, кто-то может попросить вас провести:

- *функциональное тестирование* (тщательно проверить все функции). Это говорит о том, что нужно тестировать. Вам предстоит решить, кто будет выполнять данную задачу, какие типы багов вы будете искать, как тестировать каждую функцию и как вы будете определять, есть ли в программе ошибка;
- *тестирование на экстремальные значения* (тест на ошибки, появляющиеся при вводе экстремального значения в переменную). Это подскажет вам, какой тип проблем следует искать. В данном случае вам предстоит решить, кто будет выполнять тестирование, какие переменные будут проверяться и как оценивать результаты;
- *бета-тестирование* (пригласить потребителей вашего продукта протестировать программное обеспечение). Это говорит о том, кто будет проводить тестирование. Вам предстоит решить, что рассказать этим людям (и в каком объеме), на какие части продукта обратить внимание и какие проблемы следует искать (а какие игнорировать). В одних случаях вы можете сообщить бета-тестировщикам, как распознавать определенные типы проблем, и попросить их выполнить конкретные тесты определенным образом. В других случаях вы можете позволить бета-тестировщикам самим определять, что делать и как оценивать результаты.

Техники не обязательно соответствуют только одному критерию. Они и не должны: любое тестирование сочетает в себе все пять критериев, поэтому следует ожидать, что в более полных тестовых техниках будут задействованы несколько критериев. Вот пример многомерной техники: если кто-то вас просит провести «тестирование на основе требований», то он может иметь в виду любую комбинацию из таких трех идей, как:

- покрытие (тестирование всего, что перечислено в документе с требованиями);
- потенциальные проблемы (тестирование путей, на которых эти требования могут быть не выполнены);

- оценка (разработка тестов таким образом, чтобы с помощью спецификации можно было определить, прошла программа тест или нет).

Разные тестировщики подразумевают разные комбинации этих идей, когда говорят фразу «тестирование на основе требований». Не существует единственно верной ее интерпретации[1].

Несмотря на неоднозначность (и в некоторой степени благодаря ей), мы считаем, что эта система классификации полезна в качестве генератора идей.

Если в процессе тестирования вы будете помнить обо всех пяти критериях, то можете подобрать более удачные комбинации. Как в бета-тестировании, вы можете не указывать один или несколько критериев. Вы можете не решать, как будут оцениваться результаты или как будет работать тестировщик. Однако мы предлагаем делать такой выбор осознанно, а не применять технику, ориентированную только на один из этих критериев, не понимая, что другие варианты все равно придется выбирать.

Техники, ориентированные на людей, направлены на того, кто проводит тестирование

Приведем примеры распространенных техник, которые различаются в зависимости от того, кто выполняет тестирование.

- **Пользовательское тестирование.** Тестирование с привлечением людей, которые обычно пользуются вашим продуктом. Может быть проведено в любой момент разработки, на вашем или на их компьютере, в рамках тщательно спланированных упражнений или по усмотрению пользователя. Некоторые типы пользовательского тестирования, такие как анализ задач, больше похожи на совместное исследование (с участием как минимум одного пользователя и как

[1] Многозначность понятия «тестирование на основе требований» — пример важной общей проблемы, присущей программной инженерии. Определения в нашей области изменчивы. В разных сообществах и у разных людей они используются по-разному, даже если существуют документы, которые, как можно ожидать, будут служить эталонными стандартами. Мы отложим обсуждение факторов, которые, по нашему мнению, приводят к тому, что многие люди игнорируют документы по стандартам. Отметим, что мы не претендуем на то, чтобы предложить авторитетные определения или описания методов, используемых в данной области. Одни люди будут использовать одни и те же слова для обозначения разных вещей. Другие, возможно, согласятся со смыслом нашего описания, но сформулируют его по-другому. Любая из этих позиций может быть разумной и оправданной.

минимум одного члена команды тестировщиков вашей компании), чем на тестирование, проводимое одним человеком.

- **Альфа-тестирование.** Внутреннее тестирование, выполняемое командой тестировщиков (и, возможно, другими заинтересованными сотрудниками компании).
- **Бета-тестирование.** Тип пользовательского тестирования, в котором участвуют тестировщики, которые не работают в вашей организации и являются целевой аудиторией вашего продукта. Тестируемый продукт, как правило, очень близок к стадии готовности. Многие компании считают, что любой выпуск предрелизного кода для клиентов и есть бета-тестирование, и приурочивают все бета-тесты к этапу, который называют «бета». Это ошибка. Бета-тесты действительно бывают различных видов. *Бета-тестирование дизайна*, при котором пользователям (особенно экспертам в предметной области) предлагается оценить дизайн продукта, должно проводиться как можно раньше, чтобы было время внести изменения по результатам тестирования. *Маркетинговое бета-тестирование*, призванное убедить крупных клиентов в том, что они должны купить продукт, когда он станет доступным, и установить его в своей большой сети, должно проводиться достаточно поздно, когда продукт будет довольно стабильным. При *бета-тестировании на совместимость* клиент запускает ваш продукт на аппаратной и программной платформе, которую вы не можете легко протестировать самостоятельно. Этот вид тестирования важно провести до того, как будет уже поздно находить и исправлять проблемы совместимости. Выполнение любого вида бета-тестирования требует, чтобы вы определили цели до того, как начнете планировать и проводить те или иные действия.
- **Баг-бэш.** Внутреннее тестирование с привлечением секретарей, программистов, маркетологов и всех, кто свободен. Обычно длится полдня и проводится, когда продукт уже близок к релизу. (Обратите внимание: мы приводим этот метод в качестве примера, но не одобряем его. Одни компании сочли его полезным, другие — нет.)
- **Экспертное тестирование.** Предоставьте продукт эксперту по некоторым вопросам, решаемым с помощью ПО, и получите обратную связь (описание багов, критические замечания и комплименты). Эксперт может быть или не быть тем, кто будет использовать продукт: ценность такого человека — в его знаниях, а не в принадлежности к целевой аудитории.

- **Парное тестирование.** Два тестировщика ищут баги вместе. Как правило, во время тестирования они используют один компьютер и по очереди садятся за клавиатуру.
- **Использование своего же продукта.** Ваша компания может использовать в своей работе предрелизные версии своего продукта. Прежде чем продавать ПО, она обычно ожидает, пока продукт не станет надежным настолько, чтобы его можно было реально использовать.

Техники, основанные на покрытии, направлены на то, что тестируется

Некоторые из этих техник можно классифицировать по-разному, как предметно-ориентированные, в зависимости от того, какие цели вы преследуете при их использовании. Например, интеграционное тестирование функций — метод, ориентированный на покрытие, если вы с его помощью проверяете поведение каждой функции, когда она используется в сочетании с любой другой. И он же ориентирован на решение проблем, если у вас есть теория ошибок для функций, взаимодействующих друг с другом, и вы хотите ее проверить. (Например, этот метод ориентирован на проблему, если вы хотите показать ошибки, которые возникают при передаче данных от одной функции к другой.)

В определениях, приведенных ниже и в конце главы, мы уделили дополнительное внимание тестированию предметной области, поскольку связанные с ней техники широко используются в нашей сфере и очень важны. Вы должны их знать.

- **Функциональное тестирование.** Тестирование каждой функции по очереди. Делайте это тщательно, пока не сможете с уверенностью сказать, что функция работает. Функциональное тестирование методом «белого ящика» обычно называется модульным тестированием и имеет дело с функциями в том виде, в каком вы их видите в коде. При тестировании методом «черного ящика» основное внимание уделяется командам и функциям, которые видит или может выбрать пользователь. Функциональное тестирование желательно выполнять до того, как будут проводиться более сложные тесты, затрагивающие несколько функций. В сложном тесте первая же функция с ошибкой, скорее всего, остановит его и не позволит обнаружить, что несколько других функций тоже содержат ошибки. Если полагаться на

сложные тесты, а не тестировать функции по отдельности, то это может привести к тому, что вы поздно поймете, что одна из функций не работает, потратите огромное количество сил на устранение неполадок в сложном тесте, а потом обнаружите, что проблема крылась в простой функции.

- **Интеграционное тестирование функций.** Тестирование совместной работы нескольких функций.
- **Хождение по меню.** Проход по всем пунктам меню и диалоговым окнам продукта с графическим интерфейсом пользователя, использование всех доступных вариантов.
- **Анализ предметной области (доменный анализ).** Предметная область — это (математическое) множество, содержащее все возможные значения переменной функции. При этом тестировании определяются функции и переменные. Последние могут быть входящими и исходящими (с точки зрения математики между ними разницы нет, поскольку тестовый анализ проводится в обоих случаях). Множество значений каждой переменной делится на классы эквивалентности, и из каждого класса выбирается небольшое количество примеров (обычно граничных значений). В основе техники лежит предположение, что если протестировать несколько разных представителей каждого класса, то можно найти большинство багов или все баги, которые можно было бы обнаружить при тестировании каждого члена класса. Обратите внимание: в отличие от тестирования функций основной элемент, представляющий интерес, — это переменная, а не функция. Многие переменные используются несколькими функциями. Тестировщик предметной области анализирует переменную, а затем, основываясь на данном анализе, проводит тесты, в которых эта переменная используется каждой функцией — на входе или выходе.
- **Разбиение на классы эквивалентности.** Класс эквивалентности — это набор значений переменной, которые вы считаете эквивалентными. Тестовые случаи эквивалентны, если вы верите, что: а) все они тестируют одно и то же; б) если один из них нашел баг, то и другие, скорее всего, тоже смогут это сделать; в) если один из них не нашел баг, то, вероятно, и другие его не найдут. Определив класс эквивалентности, протестируйте только один или два его члена.
- **Тестирование граничных значений.** Класс эквивалентности — это набор значений. Если вы можете отразить их на числовой оси, то граничными значениями будут наибольшие и наименьшие члены класса. При данном виде тестирования вы тестируете сами

граничные значения, а также граничные значения близлежащих классов, которые чуть меньше, чем наименьшее значение тестируемого класса, и чуть больше, чем его наибольшее значение. Например, рассмотрим поле ввода, принимающее целочисленные значения от 10 до 50. Граничные значения, представляющие интерес, — 10 (наименьшее), 9 (наибольшее целое число, которое слишком мало), 50 (наибольшее), 51 (наименьшее целое число, которое слишком велико).

- **Тестирование лучшего представителя.** Лучший представитель класса эквивалентности — это значение, которое с не меньшей вероятностью, чем любое другое значение в классе, выявляет ошибку в программном обеспечении. При тестировании граничных значений лучшими представителями всегда являются именно граничные случаи. Но предположим, что вы не можете отобразить класс эквивалентности на числовую ось. Например, принтеры, совместимые с Hewlett-Packard PCL-5, являются (или должны являться) классом эквивалентности, поскольку должны работать одинаково. Теперь предположим, что при выполнении конкретной задачи вероятность возникновения проблем у одного из принтеров несколько выше, чем у других. Этот принтер и будет лучшим представителем класса. Если он нас не подведет, то мы можем с определенной долей уверенности рассчитывать, что не подведут и остальные.
- **Список проверок или чек-лист полей ввода.** Для каждого типа поля ввода можно разработать стандартный набор тестовых примеров и повторно использовать его для аналогичных полей в этом и других продуктах. Пример данного метода мы приведем далее в этой главе (см. подраздел «Как создать чек-лист для поля ввода»).
- **Составление карты и проверка всех способов редактирования поля.** Часто значение поля можно изменить несколькими способами. Например, импортировать данные в поле, ввести их непосредственно в него, дать программе команду скопировать в поле вычисленный или пересчитанный результат и т. д. Поле имеет ограничения (на допустимые значения): постоянные или зависящие от значений других полей. Например, если J и K — беззнаковые целые числа, то они ограничены значениями от 0 до MaxInt. Это постоянные ограничения. Они зависят от того, как в языке программирования определяются целые беззнаковые числа. Предположим, что N — тоже беззнаковое целое число, N = J + K, а N = 5. В таком случае J = 5 – K, и J не может быть больше 5 (значение N). Это переменное ограничение, диапазон допустимых значений которого зависит от N. Чтобы проверить, что J находится в допустимом диапазоне (5 – K),

нужно попробовать изменить его значение, используя каждый способ ввода данных в J.

- **Логическое тестирование.** Переменные имеют взаимосвязи в программе. Например, в программе может быть правило принятия решений, которое гласит: если PERSON-AGE больше 50 и если SMOKER — это YES, то OFFER-INSURANCE должно быть равно NO. Правило принятия решений выражает логическую взаимосвязь. С помощью логического тестирования проверяются все логические связи в программе. *Построение графов причинно-следственных связей* — метод проектирования обширного набора тестов, основанных на логике.
- **Тестирование на основе состояний и переходов.** Программа переходит из состояния в состояние. В заданном состоянии одни входные данные являются допустимыми, а другие игнорируются или отклоняются. В ответ на допустимые входные данные тестируемая программа делает то, что может, и не пытается сделать то, что не должна. При тестировании на основе состояний вы прогоняете программу через большой набор переходов между состояниями и на каждом этапе тщательно проверяете результаты.
- **Тестирование путей.** Путь состоит из всех шагов, которые вы предприняли, и из всех состояний, через которые прошла программа, чтобы прийти к текущему состоянию. Тестирование путей подразумевает проверку множества путей прохождения программы. Нельзя проверить все пути в нетривиальной программе. Некоторые тестировщики проводят *тестирование подпутей*, проверяя множество частичных путей. Например, в *тестирование базовых путей* входит тестирование большинства или всех подпутей определенного типа. При этом предполагается, что если вы получите все эти пути, то несколько тестов более длинных путей, скорее всего, найдут баги, которые пропустили тесты базовых путей.
- **Покрытие операторов и ветвей.** Стопроцентное покрытие операторов достигается, если тесты выполняют каждый оператор (или строку кода) в программе. Стопроцентное покрытие операторов и ветвей достигается, если тесты выполняют каждый оператор и каждую ветвь на пути от одного оператора к другому. Проектирование тестов для достижения высокого процента покрытия строк и ветвей иногда называют тестированием на основе покрытия. (Достигнув этого показателя, можно прекратить тестирование или отказаться от разработки дополнительных тестов.) Мы называем этот метод *покрытием операторов и ветвей*, чтобы отличить от иных типов

тестирования, которые нацелены на работу с другими типами покрытия. Покрытие конфигураций — отличный пример метода, который многократно проверяет одни и те же операторы, но при этом может выдать потенциально разные результаты. Есть много других примеров (Kaner, 1995a). Для тестирования, нацеленного на достижение высоких показателей покрытия операторов и ветвей, характерно то, что оно пропускает многие виды багов, таких как (но не только) баги, связанные с отсутствием кода, некорректной обработкой граничных значений, проблемами синхронизации или совместимости с аппаратными и программными конфигурациями; трудноуловимые скрытые баги, такие как недействительные указатели, утечки памяти или повреждения стека, которые в конечном счете ведут к переполнению стека; проблемы эксплуатации и другие отказы, нарушающие требования заказчика. Этот метод гораздо более ценен как инструмент выявления неполного тестирования (какой код еще не проверялся), чем как минимальный стандарт необходимого объема тестирования. Это действительно опасно — позволять тестировщикам останавливаться только потому, что они достигли *X* процентов покрытия (Marick, 1999).

- **Покрытие конфигураций.** Если вам нужно протестировать совместимость со 100 принтерами, а вы проверили 10, то вы достигли 10%-ного покрытия. В более общем случае покрытие конфигурации измеряется процентом конфигурационных тестов, которые вы выполнили (и программа работала успешно), от общего количества запланированных конфигурационных тестов. *Почему мы называем это техникой тестирования?* Обычно мы просто считаем, что это показатель того, сколько тестов определенного типа мы провели. Однако некоторые тестировщики создают специальные серии тестов, которые позволяют ускорить и упростить тестирование большого количества конфигураций. Оптимизация усилий по достижению высокого покрытия и есть метод тестирования.
- **Тестирование на основе спецификаций.** Тестирование, направленное на проверку каждого фактического утверждения о продукте, указанного в спецификации (фактическим является любое утверждение, которое может быть показано как истинное или ложное). Сюда входит проверка каждого утверждения, приведенного в руководстве пользователя, маркетинговых документах или рекламе, а также в литературе по технической поддержке, направляемой заказчикам.

- **Тестирование на основе требований.** Тестирование, призванное доказать, что программа удовлетворяет каждому требованию, указанному в документе требований (или что некоторые требования не были выполнены).
- **Комбинированное тестирование.** Тестирование с сочетанием двух или более переменных. Мы обсудим его далее в разделе «Дополнение к техникам тест-дизайна» этой главы. Данный вид тестирования очень важен, но многие тестировщики изучают его не слишком глубоко. Большинство преимуществ, предоставляемых программой, основано на взаимодействии многих переменных. Если не варьировать их в своих тестах, то можно пропустить ошибки, вызванные сложными комбинациями, а не сложными отдельными значениями.

Проблемно-ориентированные техники направлены на причины тестирования (риски, на которые вы тестируете)

Тестирование на базе рисков имеет по меньшей мере два основных толкования.

Амланд (Amland, 1999) дал превосходное описание *менеджмента* тестирования на базе рисков. Согласно точке зрения автора данной работы, анализ рисков проводится с целью определить следующие шаги тестирования. Приоритеты тестирования определяются исходя из степени вероятности отказа некой функции программы и возможной стоимости такого отказа, если это все же произойдет. Чем выше вероятность дорогостоящего отказа, тем важнее протестировать эту функцию как можно раньше и как можно тщательнее.

Есть и другой смысл, который нам более близок: анализ рисков проводится в целях поиска ошибок. Изучая какую-либо функцию продукта, мы спрашиваем, как она может отказать. Этот вопрос распадается на множество дополнительных, например: как будет выглядеть отказ; почему эта функция должна отказать — какие факторы риска могли повлиять на нее? Мы описываем наш подход к тестированию на базе рисков в разделе «Дополнение к техникам тест-дизайна».

Оба этих подхода к данному виду тестирования рассматриваются также в книге *Risk-Based Testing* Джеймса Баха (Bach, 1999c).

В работах Уиттакера и Йоргенсена (Whittaker and Jorgensen, 1999, Whittaker and Jorgensen, 2000) представлена отличная дискуссия и при-

водятся примеры широких классов ошибок, связанных с нарушением ограничений.

- **Входные ограничения.** Ограничение — это предел того, что может обработать программа. Например, если программа может обрабатывать только 32-значные числа (или более мелкие), то программист должен предусмотреть защитные процедуры, которые будут обнаруживать входные данные, выходящие за пределы 32-значного ограничения, и отвергать их. Если такой защиты нет, то при попытке обработать подобные данные, программа потерпит неудачу.
- **Выходные ограничения.** Входные данные были корректными, но привели к выходным значениям, которые программа не может обработать. При попытке отобразить, распечатать или сохранить выходное значение программа может потерпеть неудачу.
- **Вычислительные ограничения.** Входные и выходные данные корректны, но в процессе вычисления очередного выходного значения в программе возникает сбой. Например, при перемножении двух огромных чисел произведение может быть настолько большим, что программа не сможет его вычислить.
- **Ограничения на хранение (или данные).** Входные, выходные данные и вычисления корректны, но в результате выполнения операций программа исчерпывает память или создает файлы данных, слишком большие для обработки.

Уиттакер (Whittaker, 2002) дает подробные рекомендации по тестированию с учетом этих ограничений.

Вот несколько полезных советов по проектированию тестов на базе рисков.

- Если вы проводите такое тестирование, то также должны выполнить сопоставимое тестирование, не основанное на рисках, чтобы проверить, насколько велика вероятность, что вы не знаете риски настолько хорошо, чтобы принять правильные решения.
- Проводите тестирование на синхронизацию. Удивительно, но многие тестировщики, получившие американское образование, не учитывают ее. К классическим проблемам синхронизации относятся условия гонки и другие неожиданные последовательности событий, происходящих во времени.
- При создании теста всегда создавайте процедуру, которая заставит программу использовать введенные вами тестовые данные, что позволит определить, не использует ли она их неправильно.

Техники тестирования, основанные на подходах к тестированию

- **Регрессионное тестирование.** Данная техника подразумевает повторное использование одних и тех же тестов, что позволяет проводить повторное тестирование (с их помощью) после внесения изменений. Существует три вида регрессионного тестирования[1]. *Баг-ретест* проводится после отправки отчета о баге и получения ответа, что баг исправлен. Цель — доказать, что баг не был исправлен. *Регрессионное тестирование старых багов* проводится с целью доказать, что изменение в программном обеспечении привело к возврату старого бага. *Регрессионное тестирование побочных эффектов*, называемое также *стабилизацией*, предполагает повторное тестирование существовавших ранее частей продукта. Цель — доказать, что изменение привело к тому, что компоненты, которые раньше работали, теперь не работают.
- **Тестирование по сценарию.** Ручное тестирование, обычно выполняемое младшим тестировщиком, который следует пошаговой процедуре, написанной более старшим тестировщиком.
- **Смоук-тестирование.** Этот тип тестирования проводится с целью доказать, что новую сборку не стоит тестировать. Смоук-тесты часто автоматизированы и стандартизируются по мере проверок сборок. Тесты проверяют те компоненты, которые, как вы ожидаете, будут работать, и если они не функционируют, то это может служить сигналом, что программа была собрана с неправильным файлом или что-то перестало работать.
- **Исследовательское тестирование.** Мы ожидаем, что на протяжении всего проекта тестировщик будет узнавать о продукте, его рынке, рисках и о причинах неудач в предыдущих. Постоянно создаются и используются новые тесты. Они более эффективны, чем старые, поскольку основаны на постоянно пополняющихся знаниях тестировщика.
- **Партизанское тестирование.** Быстрая и жестокая атака на программу. Форма исследовательского тестирования, которое обычно ограничено по времени и проводится опытным исследователем. Например, старший тестировщик может потратить день на про-

[1] С момента выхода оригинала книги взгляд на регрессионное тестирование изменился. Сейчас регрессионным тестированием в основном называют последний из перечисленных здесь видов. А стабилизацией называют последовательное выполнение нескольких итераций регрессионного тестирования с определенными критериями выхода (окончания) стабилизации. — *Примеч. науч. ред.*

верку области, которая в противном случае была бы проигнорирована. Он пробует свои самые мощные атаки. И если находит серьезные проблемы, то на доработку этой области будут выделены дополнительные ресурсы и время, что может повлиять на общий план тестирования. Если он не обнаружит существенных проблем, то в дальнейшем эта область будет игнорироваться или подвергаться лишь поверхностному тестированию.

- **Тестирование на основе сценариев использования.** По нашему мнению, обычно соответствует четырем критериям. 1. Тест должен быть реалистичным. Он должен повторять действия, которые действительно будут выполнять заказчики. 2. Тест должен быть сложным, содержать несколько сложных для программы функций. 3. Тест позволяет легко и быстро определить, прошла его программа или нет. 4. Стейкхолдер, скорее всего, будет активно доказывать, что программу следует исправить, если она не прошла эту проверку. Тест, соответствующий всем четырем критериям, будет убедительным и, вероятно, позволит исправить баги, если они приведут к сбою программы. Однако возможно, что на разработку подобного теста придется потратить несколько дней. Тесты, основанные на сценариях использования, нередко также называют *бизнес-тестированием* (Jacobson, 1992; Collard, 1999). (Многие классифицируют их как тесты, основанные на покрытии, потому что они уделяют особое внимание важным сценариям использования.)
- **Тестирование инсталляции.** Установите программное обеспечение различными способами и на разные типы систем, на которые оно может быть установлено. Проверьте, какие файлы добавлены и изменены на диске. Работает ли установленное ПО? Что происходит при его удалении?
- **Нагрузочное тестирование.** Программа или система подвергается атаке при запуске в системе, которая предъявляет множество требований к ресурсам. При достаточно высокой нагрузке система, вероятно, откажет, но картина событий, приведших к отказу, укажет на уязвимости в проверяемой программе или системе, которые могут быть использованы при более нормальном использовании тестируемого ПО. Книга Асбока (Asbock, 2000) содержит вводную информацию о нагрузочном тестировании.
- **Тестирование длинных последовательностей.** Проводится в течение ночи либо нескольких дней или недель. Цель — обнаружить ошибки, которые короткие последовательности тестов не заметят. Примеры ошибок, которые часто находят таким образом, — указатели на несуществующие объекты, утечки памяти, переполнение

стека, плохое взаимодействие нескольких функций. (Эту технику иногда называют тестированием на продолжительность, тестированием на надежность или тестированием на выносливость.)

- **Тестирование производительности.** Эти тесты проводят для определения скорости работы программы, чтобы решить, нуждается ли она в оптимизации. Но они могут выявить и многие другие баги. Значительное изменение производительности по сравнению с предыдущим выпуском может свидетельствовать о наличии ошибки в коде. Например, если вы проверяете, сколько времени занимает выполнение простого функционального теста сегодня, а затем запу́стите этот же тест на той же машине завтра, то, скорее всего, уточните у программиста или напишете баг-репорт, если тест будет выполняться более чем в три раза быстрее или медленнее. В любом случае это подозрительно, поскольку в программе было изменено что-то фундаментальное[1].

Техники тестирования, основанные на оценке, направлены на то, как вы оцениваете результаты теста

Техники, основанные на оценке, описывают способы, с помощью которых можно определить, прошла ли программа тест. Они не дают указаний о том, как должно проводиться тестирование или как должны собираться данные. Речь идет о том, что если вы можете собрать определенные данные, то можете их оценить.

- **Самопроверяющиеся данные.** Файлы данных, используемые при тестировании, содержат информацию, позволяющую определить, повреждены ли выходные данные.
- **Сравнение с эталонными результатами.** Регрессионное тестирование (обычно, но не всегда автоматизированное), при котором успех или провал определяется путем сравнения результатов, полученных сегодня, с результатами прошлой недели. Если на прошлой неделе результат был правильным, а сейчас он другой, то разница может свидетельствовать о новом дефекте.

---

[1] Сэм Гакенхаймер отметил: «Разница в производительности может также означать изменения в сторонних компонентах или конфигурации. Например, изменения в JVM в различных выпусках JDK от Sun привели к значительному изменению характеристик производительности. Поскольку это компонент, обновляемый заказчиком, тестирование производительности может дать неожиданные результаты, даже если ваш код совсем не изменился!»

- **Сравнение со спецификацией или другим авторитетным документом.** Несоответствие спецификации, вероятно, означает ошибку.
- **Эвристическое соответствие.** Соответствие — важный критерий оценки программы. Несоответствие может послужить причиной для составления баг-репорта или свидетельствовать о намеренном изменении архитектуры программы. Мы работаем с семью основными видами соответствия.
  1. *Соответствие истории.* Нынешнее поведение функции соответствует ее прошлому поведению.
  2. *Соответствие представлению.* Поведение функции соответствует представлению, которое хочет создать организация.
  3. *Соответствие аналогичным продуктам.* Поведение функции соответствует поведению аналогичных функций в продуктах-аналогах.
  4. *Соответствие требованиям.* Функция ведет себя ожидаемым образом.
  5. *Соответствие ожиданиям пользователей.* Поведение функции соответствует тому, чего, по нашему мнению, хотят пользователи.
  6. *Соответствие продукту.* Поведение функции соответствует поведению сопоставимых функций или функциональных шаблонов проектирования внутри продукта.
  7. *Соответствие целям.* Функция ведет себя согласно своему прямому назначению.
- **Тестовый оракул.** Оракул — это инструмент оценки, который позволяет определить, прошла ли программа тест. Если тестирование крупномасштабное и автоматизированное, то оракулом, вероятно, является другая программа, которая генерирует результаты или проверяет результаты тестируемого ПО. Оракулу обычно больше доверяют, чем тестируемому программному обеспечению, поэтому выявленная им проблема, как правило, стоит времени и усилий, потраченных на ее проверку.

Классификация техники зависит от того, как вы о ней думаете

Возможно, вы недоумеваете, почему мы распределили техники по урокам именно таким образом. Напоминаем: любое тестирование содержит все пять аспектов пятимерной системы. Мы перечислили техники по категориям просто для того, чтобы дать вам представление о том, как в разных методах одни способы мышления превалируют над другими.

Ваше мнение может быть другим. Например, один из читателей утверждал, что нагрузочное тестирование следует классифицировать как проблемно-ориентированное (или риско-ориентированное), а не как тестирование, основанное на виде деятельности. Мы ответили, что можно думать и так и так.

Посмотрим на это в проблемно-ориентированном ключе.

- Нагрузочное тестирование можно рассматривать с точки зрения эффекта атак типа «отказ в обслуживании». Злоумышленник может попытаться отказать в обслуживании, создав слишком много подключений или пользователей, либо используя слишком много памяти (заставив каждого пользователя одновременно выдать команду, требующую больших объемов памяти), либо ставя задачи, на которые уходит слишком много вычислительной мощности. Для каждого из этих типов риска можно провести различные нагрузочные тесты.

Теперь рассмотрим то же самое с точки зрения видов активности.

- Используйте инструмент для отслеживания моделей поведения ваших клиентов. Какие команды они используют чаще всего? Какие задачи чаще всего пытаются решить? Какой процент клиентов проявляет ту или иную активность? Когда у вас появится модель поведения пользователей на вашем сайте, приобретите инструмент нагрузочного тестирования и запрограммируйте его на сценарии, похожие на каждый из типов использования. Пусть этот инструмент случайным образом выбирает сценарии — по сути, создает различные сессии, представляющие разные типы пользователей. Продолжайте добавлять сессии и наблюдайте за тем, как производительность и надежность системы снижается при увеличении нагрузки. По необходимости внесите изменения в программное обеспечение.

Когда вы смотрите на программу с точки зрения рисков, вы думаете о ее слабых местах и спрашиваете, как спроектировать тесты, которые выявят такие места. Когда вы знаете, какой тип теста хотите провести, подумайте о том, как вы будете это делать. Если вы тестируете телефонную систему, то можете использовать инструмент или привлечь десять друзей, которые совершат множество звонков. Критерий проектирования теста — его эффективность: он должен обнаруживать сбой, который вы ищете.

Напротив, когда вы смотрите на задачу с точки зрения действий, вы спрашиваете, как проводить нагрузочное тестирование. С помощью каких инструментов? Что они будут делать? И т. д. Предполагается, что

если вы грамотно используете инструменты, составляете точную модель поведения клиентов и выполняете другие действия, связанные с качественным нагрузочным тестированием, то, вероятно, найдете те типы багов, которые способно выявить нагрузочное тестирование.

Любая из этих классификаций точна, но сама по себе она поможет вам лишь в малой степени. Как бы вы ни классифицировали такую технику, как нагрузочное тестирование, когда придет время тестировать, перед вами будут стоять все те же пять вопросов, пять критериев, влияющих на принятие решения.

1. *Кто будет проводить тестирование?*
2. *Какие аспекты программы будут тестироваться?*
3. *Какие типы проблем вы будете искать?*
4. *Какие конкретно задачи вы будете решать при тестировании?*
5. *Как вы поймете, был тест успешным или нет?*

ДОПОЛНЕНИЕ К ТЕХНИКАМ ТЕСТ-ДИЗАЙНА

Ниже приведены более подробные описания нескольких ключевых техник тестирования, которые нам показались особенно полезными.

- Как создать чек-лист для поля ввода.
- Как создать чек-лист для повторяющихся проблем.
- Как создать таблицу трассировки требований для тестирования на основе спецификаций.
- Как проводить комбинированное тестирование с помощью техники попарного тестирования.
- Как проанализировать риски, связанные с тем или иным элементом или аспектом программы.

Как создать чек-лист для поля ввода

Начните с вопроса: «Что было бы интересно проверить при тестировании простого целочисленного поля ввода?» Вот некоторые ответы, которые мы считаем обычными для такого поля.

- Отсутствие ввода.
- Пустое поле (очистить значение по умолчанию).
- Значение с количеством цифр или символов больше допустимого.

- Нуль.
- Допустимое значение.
- Значение нижней границы – 1.
- Значение нижней границы.
- Значение верхней границы.
- Значение верхней границы + 1.
- Значение гораздо ниже нижней границы.
- Значение гораздо выше верхней границы.
- Значение с минимально допустимым количеством цифр или символов.
- Значение с количеством цифр или символов на 1 меньше минимально допустимого.
- Значение с максимально допустимым количеством цифр или символов.
- Значение с количеством цифр или символов на 1 больше максимально допустимого.
- Значение с количеством цифр или символов значительно больше максимально допустимого.
- Отрицательные числа.
- Не цифры, особенно / (символ ASCII 47) и : (символ ASCII 58).
- Неправильный тип данных (например, десятичная дробь в целом числе).
- Выражения.
- Начальный пробел.
- Много начальных пробелов.
- Начальный нуль.
- Много начальных нулей.
- Начальный знак +.
- Много начальных знаков +.
- Непечатаемый символ (например, Ctrl+символ).
- Зарезервированные и шаблонные символы, распознаваемые файловой системой (ФС) (например, \ * . :).
- Зарезервированные символы языка.
- Символы из верхнего диапазона ASCII (128–254) (также известные как ANSI).

- ASCII 255 (часто интерпретируется как конец файла).
- Символы в верхнем регистре.
- Символы в нижнем регистре.
- Модификаторы (например, Ctrl, Alt, Shift-Ctrl и т. д.).
- Функциональная клавиша (F2, F3, F4 и т. д.).
- Ничего не вводите, а долго ждите, прежде чем нажать клавишу Enter или Tab, щелкнуть на OK или сделать что-то подобное, чтобы поле потеряло фокус ввода. Существует ли тайм-аут? Каков его эффект?
- Введите одну цифру, но подождите долгое время, прежде чем продолжить ввод, а затем нажмите клавишу Enter. Сколько придется ждать, пока система прервет ввод по тайм-ауту? Что произойдет с введенными вами данными? Что произойдет с другими данными, которые вы ввели ранее?
- Введите цифры, отредактируйте их с помощью клавиши Backspace, удалите и используйте клавиши со стрелками (или мышь), чтобы перейти к уже введенным цифрам и вставить или набрать новые цифры.
- Вводите цифры в то время, когда система реагирует на прерывания различного рода (такие как работа принтера, события часов, движение мыши и щелчки ее кнопками, перемещение файлов на диск и т. д.).
- Введите цифру, переключитесь на другое приложение, вернитесь к этому приложению. Где находится фокус ввода?

Подобный список часто называют *списком тестов* (Марик (Marick, 1995) дает определение этому термину и приводит примеры). Мы считаем полезным представить этот список в форме чек-листа, как в табл. 3.1.

В верхней части чек-листа перечислены тесты, которые вы будете использовать снова и снова. Сбоку — поля, которые вы будете тестировать. Например, в типичном диалоговом окне Print (Печать) одним из полей является Number of Copies (Количество копий). Диапазон допустимых значений для данного поля обычно составляет от 1 до 99 или от 1 до 255 (в зависимости от принтера). В форме можно написать `Печать: Количество копий` в одной строке, затем выполнить несколько тестов для этого поля (или все) и потом зафиксировать результаты. (Нам нравится использовать зеленый и розовый маркеры для закрашивания ячеек, в которых были получены положительные и отрицательные результаты тестирования соответственно.)

Таблица 3.1. Чек-лист испытаний полей числового ввода

Поле ввода числовых данных	Отсутствие ввода	Пустое поле (очистить значение по умолчанию)	Ноль	Нижняя граница – 1	Нижняя граница	Верхняя граница	Верхняя граница + 1	Гораздо ниже нижней границы	Гораздо выше верхней границы	Максимальное количество символов верхней границы	Максимальное количество символов + 1	Намного больше максимального количества символов	Отрицательные числа	Не цифры (/ ASCII 47)	Не цифры (: ASCII 58)	Неправильный тип данных	Выражения	Начальные пробелы	Непечатаемый символ	Шаблонные символы ФС	Верхние символы ASCII	Верхний регистр	Нижний регистр	Модификаторы (Ctrl, Alt и т. д.)	Функциональные клавиши

Чек-лист определяет структуру для оформления заданий. При добавлении новой функции или изменении существующей вы можете передать несколько таких стандартных чек-листов тестировщику, который относительно недавно работает в проекте (но имеет опыт тестирования). Его задача — проверить работу новой функции или корректность старых функций, на которые может повлиять измененная функция.

Поле ввода целого числа — лишь один из примеров. Возможно, вы посчитаете полезным создание подобных схем для рациональных чисел (разной точности), символьных полей (различной ширины), имен файлов, их местоположения, дат и т. д. Если вы то и дело сталкиваетесь с одним типом полей ввода в разных программах или внутри одной тестируемой программы, то стоит потратить время на создание многократно используемого чек-листа.

Как создать чек-лист для повторяющихся проблем

Чек-лист для поля ввода — лишь один из примеров широкого класса полезных чек-листов, которые можно создать. Поля ввода не единственные кандидаты на стандартизацию. Если какая-либо ситуация часто повторяется на стыке программ и внутри них, то у вас есть основания потратить время и силы на создание списка тестов. Такой список всегда можно представить в виде чек-листа.

Приведем пример, в котором не используются входные переменные.

В этом случае в списке перечислены возможные варианты неудачных попыток программы записать файл на диск. В некоторых ситуациях программа попытается записать файл, например:

- сохранение нового файла;
- перезапись файла с таким же именем;
- добавление в файл;
- замена редактируемого файла новой версией с тем же именем;
- экспорт в новый формат файла;
- печать на диск;
- запись сообщений или ошибок на диск;
- сохранение временного файла (многие программы делают это автоматически, поэтому при тестировании пользовательского интерфейса об этом можно не думать. Однако если диск переполнен, программа все равно может дать сбой).

Каждая из этих ситуаций будет иметь свою строку в списке. Аналогично если тестируемое программное обеспечение позволяет экспортировать данные в различные форматы, то чек-лист будет включать строки по одной на каждый формат.

В столбцах указаны выполняемые тесты. Например, попробуйте сохранить файл на заполненный диск. Попробуйте сохранить файл на почти заполненный диск. Попробуйте сохранить файл на диск, который отключился, и т. д.

Вот список некоторых интересных тестовых случаев неудачных попыток сохранения файла.

- Сохранение на заполненный локальный диск.
- Сохранение на почти заполненный локальный диск.
- Сохранение на локальный диск, защищенный от записи.
- Сохранение на заполненный диск в локальной сети.
- Сохранение на почти заполненный диск в локальной сети.
- Сохранение на защищенный от записи диск в локальной сети.
- Сохранение на заполненный диск в удаленной сети.
- Сохранение на почти заполненный диск в удаленной сети.
- Сохранение на защищенный от записи диск в удаленной сети.
- Сохранение в файл, каталог или на диск при отсутствии права на запись.
- Сохранение на поврежденный (ошибка ввода-вывода) локальный диск, диск в локальной или удаленной сети.
- Сохранение на неформатированный локальный диск, диск в локальной или удаленной сети.
- Удаление локального или сетевого диска из привода после открытия файла.
- Измерение тайм-аута ожидания повторного появления локального или сетевого диска.
- Активное взаимодействие с клавиатурой и мышью в процессе сохранения на локальный или сетевой диск.
- Генерация какого-либо другого прерывания во время сохранения на локальный или сетевой диск.
- Выключение питания (локального компьютера) во время сохранения на локальный или сетевой диск.
- Отключение питания (диска или компьютера, подключенного к диску) во время сохранения на локальный или сетевой диск.

Чтобы создать такой список, мы предлагаем вам провести два мозговых штурма с коллегами. Во время первого постарайтесь вспомнить обо всем, что могло бы стать тестом, который вы бы регулярно проводили

для проверяемого объекта (например, поле ввода) или задачи (например, сохранение файла). Выделите час времени, заполните много флипчартов, а затем пусть ваши коллеги на день вернутся к работе, пока вы будете систематизировать материал, полученный в ходе мозгового штурма.

Чтобы систематизировать материал, создайте новый набор страниц флипчарта. На каждой напишите тематический заголовок, например «Объем диска» или «Прерывание во время записи». Под заголовком скопируйте все элементы с флипчартов, которые соответствуют этой теме. В итоге все элементы окажутся на одном из тематических флипчартов или будут отбракованы (не стесняйтесь отказываться от глупых идей).

На следующий день проведите мозговой штурм, используя тематические таблицы. Люди будут добавлять новые пункты в темы «объем диска», «прерывание во время записи» и т. д. Подготовьте несколько запасных таблиц для новых тем. Нередко на второй встрече список удваивается.

После второй встречи рассортируйте тесты на важнейшие, которые войдут в основной чек-лист; редко используемые, которые войдут в дополнительный список, который можно распространить вместе с основным чек-листом; и отбракованные.

В работе Нгуена (Nguyen, 2000) приводятся дополнительные примеры чек-листов.

Как создать таблицу трассировки требований на основе спецификаций

Таблица трассировки требований позволяет отслеживать каждый тестовый случай до элемента (элементов) спецификации и обратно от каждого элемента спецификации до всех тестовых случаев, в которых он проверяется. В табл. 3.2 приведен пример.

Каждый столбец содержит отдельный элемент спецификации, который может иметь отношение к функции, переменной, значению (например, граничному случаю) переменной, обещанной выгоде, предположительной совместимости с устройством или любому другому утверждению, истинность или ложность которого можно доказать.

Каждая строка представляет собой тестовый случай.

В каждой ячейке показано, с помощью какого тестового случая проверяются те или иные элементы.

Таблица 3.2. Таблица трассировки требований

Тестовый случай	**Элемент спецификации**					
	1	**2**	**3**	**4**	**5**	**6**
1	X	—	X	—	—	X
2	X	X	—	X	—	X
3	—	—	X	X	—	X
4	—	—	X	X	—	X
5	X	—	—	—	X	X
6	—	X	—	—	—	X
ИТОГО	3	2	3	3	1	6

Если функция изменится, то можно быстро увидеть, какие тесты необходимо проанализировать повторно и, возможно, переписать. В целом можно проследить путь от интересующего элемента до тестов, которые его покрывают.

Эта таблица не является идеальным тестовым документом. Она не описывает тесты, а лишь сопоставляет тест-кейсы с элементами спецификации. Она не позволяет увидеть, сильный тест или слабый, делает ли он с функцией (или другим элементом спецификации) что-то интересное или такое, что никого не интересует. Вы также не можете увидеть, как тестируются функции, которые не были указаны, или поправки, которые вы внесли в процессе тестирования, чтобы изменить некорректные спецификации. Несмотря на эти проблемы, подобные таблицы могут помочь заметить, что:

- одна функция (или элемент) почти никогда не тестируется, а другая тестируется очень часто;
- изменение одного элемента (например, номер 6 в табл. 3.2) приведет к пересмотру огромного количества тестов в системе. (Это ключевой вопрос при контрактной разработке, поскольку заказчик заплатит большие деньги за тестирование, если получит то изменение, которое запрашивает, и его следует предупредить об этом до внесения изменения.)

Таблицы трассировки полезны не только при тестировании на основе спецификаций. Определив список целей для тестирования (элементов спецификации, функций, вариантов использования, сетевых карт и т. д.), вы можете разместить цели по столбцам, а тестовые случаи по строкам и затем проверить, какие тестовые случаи что проверяют. Так

вы почти наверняка найдете огрехи в своем тестировании. Если тестирование автоматизировано, то, возможно, чек-лист удастся сгенерировать автоматически.

Как проводить комбинированное тестирование с помощью техники попарного тестирования

Комбинированное тестирование предполагает совместное тестирование нескольких переменных. Первая критическая проблема комбинированного тестирования — количество тестовых случаев. Представьте совместное тестирование трех переменных, когда каждая из них имеет 100 возможных значений. Количество возможных совместных тестовых случаев равно 100 × 100 × 100 = 1 000 000. Сокращение количества тестов — важнейшая задача.

Начните с разбиения предметной области

Первый шаг — сократить количество тестируемых значений каждой переменной. Наиболее распространенный подход заключается в тестировании предметной области. Разделите значения переменной 1 на подмножества и выберите лучших представителей этих подмножеств. Возможно, таким образом удастся сократить количество тестов переменной 1 до пяти. Если вы сможете сделать то же самое для переменных 2 и 3, то теперь у вас будет всего 5 × 5 × 5 = 125 тестов. Для применения на практике это все равно слишком много, но гораздо меньше, чем миллион.

Лучше всего вопросы разделения рассмотрены в работах Остранда и Бальцера (Ostrand, Balcer, 1988), а также Йоргенсена (Jorgensen, 1995). Последний приводит хорошие примеры разделения и совместного тестирования нескольких разделенных переменных. Мы представляем другой подход к комбинациям, который отличается от подхода Йоргенсена, и считаем его полезным.

Достижение всех одиночек

Простейший набор комбинированных тестов позволяет охватить все интересующие вас значения (каждое из пяти, которые мы будем тестировать) каждой переменной. Это называется «все одиночки» (в отличие от «всех пар» и «всех троек»), поскольку вы должны убедиться, что охвачено каждое значение каждой переменной. Этого можно добиться следующим образом.

1. Пусть V1, V2 и V3 обозначают три переменные.
2. Пусть A, B, C, D и E — пять интересующих нас значений переменной V1. В частности, предположим, что V1 — это операционная система; A — Windows 2000; B — Windows 95; C — Windows 98 original; D — Windows 98 с первым пакетом обновления; E — Windows ME.
3. Пусть I, J, K, L и M — пять интересующих нас значений переменной V2. В частности, предположим, что V2 — это браузер; I — Netscape 4.73; J — Netscape 6; K — Explorer 5.5; L — Explorer 5.0; M — Opera 5.12 для Windows.
4. Пусть V, W, X, Y и Z — пять интересующих нас значений переменной V3. Они относятся к пяти различным видам дисковых накопителей в системе.

Чтобы проверить все комбинации значений этих переменных, нам потребуется 5 × 5 × 5 = 125 испытаний.

Таблица 3.3 представляет собой таблицу комбинированных тестов, которая обеспечивает «полное тестирование», когда критерием полноты является то, что каждое значение каждой переменной должно присутствовать хотя бы в одном тесте.

Таблица 3.3. «Все одиночки» — все значения представлены хотя бы один раз

Тестовый случай	**Переменная 1**	**Переменная 2**	**Переменная 3**
1	A (Win 2K)	I (Netscape 4.73)	V (разновидность диска 1)
2	B (Win 95)	J (Netscape 6)	W (разновидность диска 2)
3	C (Win 98)	K (IE 5.5)	X (разновидность диска 3)
4	D (Win 98 SP1)	L (IE 5.0)	Y (разновидность диска 4)
5	E (Win ME)	M (Opera 5.12)	Z (разновидность диска 5)

Такой подход часто используется в конфигурационном тестировании для сокращения количества тестируемых конфигураций до управляемого числа.

Серьезная проблема этого подхода заключается в том, что он упускает предсказуемо важные конфигурации. Например, многие люди могут использовать Explorer 5.5 с Windows ME, но этот тест в табл. 3.3 не указан. Зато в ней есть комбинация Opera 5.12 с Windows ME.

Обычное решение этой проблемы — указать дополнительные тестовые случаи, содержащие ключевые пары переменных (например, Explorer 5.5 с Windows ME) или ключевые комбинации из нескольких переменных (например, Explorer 5.5, Windows ME, принтер HP 4050N, 256 Мбайт оперативной памяти и 21-дюймовый цветной монитор с разрешением 1600 × 1200). Сотрудники отдела маркетинга или технической поддержки могут указать эти варианты, а также, возможно, 10 или 20 дополнительных ключевых комбинаций, в общей сложности 15 или 25 тестов.

Достижение всех пар

При подходе *«все пары»* (Cohen et al., 1996 и 1997) набор тестовых случаев содержит все пары значений каждой переменной. Так, E (Windows ME) не просто образует пару с M (Opera). Она также сочетается с I, J, K и L. Аналогично E сочетается с каждым значением V3.

В табл. 3.4 приведен набор комбинаций, удовлетворяющих критерию «все пары». Каждое значение каждой переменной сочетается с каждым значением любой другой переменной как минимум в одном тестовом случае. Этот стандарт является гораздо более строгим, чем «все одиночки», но позволяет существенно сократить количество тестовых случаев: со 125 (все комбинации) до 25.

Чтобы показать, как создать набор тестов, состоящий из всех пар, рассмотрим более простой пример.

Пошаговый пример

Представьте программу с тремя переменными: V1 имеет три возможных значения; V2 — два возможных значения; V3 — два возможных значения. Если V1, V2 и V3 независимы, то количество возможных комбинаций: 3 × 2 × 2 = 12.

Чтобы создать таблицу всех пар, выполните следующие действия.

1. Присвойте столбцам имена переменных, перечисляя их в порядке убывания (по количеству возможных значений).
2. Если переменная в столбце 1 имеет V1 возможных значений, а переменная в столбце 2 — V2 возможных значений, то в таблице будет не менее V1 × V2 строк (нарисуйте таблицу таким образом, но оставьте пустую строку или две между группами повторений в столбце 1).
3. Заполните таблицу, по одному столбцу за подход. В первом столбце каждый из элементов повторяется V2 раз, пропускается строка,

а затем начинается повторение следующего элемента. Например, если возможные значения переменной 1 — A, B и C, а V2 равно двум, то столбец 1 будет содержать A, A, пустая строка, B, B, пустая строка, C, C, пустая строка. Пропустите пустую строку, поскольку трудно определить, сколько тестов (сколько строк) потребуется. Оставьте место для дополнительных тестов.

Таблица 3.4. Все пары значений представлены хотя бы один раз (25 тестов вместо 125)

Тестовый случай	**Переменная 1**	**Переменная 2**	**Переменная 3**
1	A	I	V
2	A	J	W
3	A	K	X
4	A	L	Y
5	A	M	Z
6	B	I	W
7	B	J	Z
8	B	K	Y
9	B	L	V
10	B	M	X
11	C	I	X
12	C	J	Y
13	C	K	Z
14	C	L	W
15	C	M	V
16	D	I	Y
17	D	J	X
18	D	K	V
19	D	L	Z
20	D	M	W
21	E	I	Z
22	E	J	V
23	E	K	W
24	E	L	X
25	E	M	Y

4. Во втором столбце перечислите все значения переменной, пропустите строку, перечислите значения и т. д. Например, если возможными значениями переменной 2 являются X и Y, то в данный момент таблица выглядит так (табл. 3.5).

Таблица 3.5. Первый шаг создания таблицы всех пар

Переменная 1	**Переменная 2**	**Переменная 3**
A	X	
A	Y	
B	X	
B	Y	
C	X	
C	Y	

5. Добавьте третий столбец (третью переменную).

   Каждый раздел третьего столбца (считайте, что две строки AA определяют один раздел, BB — другой и т. д.) должен содержать все значения переменной 3. Упорядочите значения таким образом, чтобы переменные также составляли все пары с переменной 2.

   Предположим, переменная 3 может иметь значения 0 или 1. Третий раздел может быть заполнен любым способом, и вы можете выделить цветом свой выбор в таблице, чтобы впоследствии при необходимости отменить его. Решение (скажем, 1,0) является произвольным (табл. 3.6).

Таблица 3.6. Второй шаг создания таблицы всех пар

Переменная 1	**Переменная 2**	**Переменная 3**
A	X	1
A	Y	0
B	X	0
B	Y	1
C	X	1
C	Y	0

Теперь, когда задача с тремя столбцами решена, попробуйте добавить больше переменных. Каждая из них будет иметь два значения.

Чтобы добавить переменную с несколькими значениями, вам придется начать все сначала, поскольку порядок переменных в таблице должен быть таким: от переменной с наибольшим количеством значений к следующей по величине и далее вниз, чтобы в последнем столбце была переменная с наименьшим количеством значений. (Вы *можете* сделать это и по-другому, но, как показывает наш опыт, совершите столько ошибок, что пытаться сделать это иначе было бы неразумно.)

Четвертый столбец добавится легко. Сначала убедитесь, что охватили все пары значений в столбцах 4 и 2 (это можно сделать в блоках AA и BB), а затем — что охватили все пары значений в столбцах 4 и 3 (табл. 3.7).

Таблица 3.7. Добавление четвертой переменной в таблицу всех пар

Переменная 1	**Переменная 2**	**Переменная 3**	**Переменная 4**
A	X	1	E
A	Y	0	F
B	X	0	F
B	Y	1	E
C	X	1	F
C	Y	0	E

Посмотрите на первую попытку в столбце 5 (табл. 3.8). Она включает пары GH для всех комбинаций значений в столбцах 1, 2 и 3, но не включает их для всех комбинаций в столбце 4.

Последним произвольным выбором был HG в разделе BB. (После того как был определен порядок сначала H, затем G в разделе BB, HG — необходимый порядок для третьего раздела, чтобы поставить H в пару к 1 в третьем столбце.)

Чтобы не допустить неправильного предположения, что HG — подходящий порядок для второго раздела, очистите его и попробуйте снова.

Таблица 3.8. Добавление пятой переменной в таблицу всех пар (этот вариант не работает, но показывает, как сделать предположение, а затем восстановить его, если предположение неверно)

Переменная 1	Переменная 2	Переменная 3	Переменная 4	Переменная 5
A	X	1	E	G
A	Y	0	F	H
B	X	0	F	H
B	Y	1	E	G
C	X	1	F	H
C	Y	0	E	G

6. Измените последний произвольный выбор (столбец 5, раздел BB, вместо HG — GH).
7. Сотрите раздел CC, поскольку выбор HG в нем был основан на том, что предыдущий раздел содержал HG, а мы его только что очистили.
8. Заполните раздел CC, проверив наличие недостающих пар. GH, GH дадут две пары XG, XG, поэтому запишите HG в третьем разделе. В результате получается столбец 2X со столбцом 5H и столбец 2Y со столбцом 5G, что необходимо для получения всех пар (табл. 3.9).

Таблица 3.9. Успешное добавление пятой переменной в таблицу всех пар

Переменная 1	Переменная 2	Переменная 3	Переменная 4	Переменная 5
A	X	1	E	G
A	Y	0	F	H
B	X	0	F	G
B	Y	1	E	H
C	X	1	F	H
C	Y	0	E	G

Если попытаться добавить еще одну переменную, то она не впишется в шесть пар. Попробуйте сделать это с IJ (значениями переменной 6) в любом порядке, и ничего не получится (табл. 3.10).

Таблица 3.10. Эти шесть переменных не вписываются в шесть тестов в таблице всех пар

Переменная 1	**Переменная 2**	**Переменная 3**	**Переменная 4**	**Переменная 5**	**Переменная 6**
A	X	1	E	G	I
A	Y	0	F	H	J
B	X	0	F	H	J
B	Y	1	E	G	I
C	X	1	F	H	J
C	Y	0	E	G	I

Переменная 1	**Переменная 2**	**Переменная 3**	**Переменная 4**	**Переменная 5**	**Переменная 6**
A	X	1	E	G	I
A	Y	0	F	H	J
B	X	0	F	H	I
B	Y	1	E	G	J
C	X	1	F	H	J
C	Y	0	E	G	I

Однако это легко исправить. Нужно всего лишь добавить еще два тестовых случая (табл. 3.11). Взглянув на вторую таблицу (см. табл. 3.10), нетрудно заметить, что нужны еще два теста: один с парой G и J и второй с парой H и I. Значения остальных переменных неважны (в смысле охвата всех пар), поэтому заполняйте их чем угодно. Если вы собираетесь продолжать добавлять переменные, то можете оставить их пустыми, а позже (когда попытаетесь вместить переменные 7 и 8 в те же восемь тестовых случаев) решить, какие значения были бы уместны в этих строках.

Если бы мы попытались проверить все комбинации этих переменных, то получилось бы 3 × 2 × 2 × 2 × 2 × 2 × 2 = 96 тестов. Мы существенно сократили набор тестов, использующих все пары, — с 96 до 8.

Таблица 3.11. Все пары с шестью переменными в восьми тестовых примерах

Переменная 1	Переменная 2	Переменная 3	Переменная 4	Переменная 5	Переменная 6
A	X	1	E	G	I
A	Y	0	F	H	J
				G	J
B	X	0	F	G	I
B	Y	1	E	H	J
				H	I
C	X	1	F	H	J
C	Y	0	E	G	I

Если вы будете использовать *только* все пары, это может быть чревато. Как и в случае со «всеми одиночками», вы можете знать конкретную комбинацию, которая широко используется или может вызвать затруднения. Лучше всего добавить этот случай в таблицу. Вы сократили количество тестов с 96 до 8. Целесообразно расширить набор до 10 или 15 тестов, чтобы покрыть важные специальные случаи. Другой рабочий пример приведен в работе Коэна (Cohen et al., 1997).

Как проанализировать риски, связанные с тем или иным элементом или аспектом программы

Предположим, вы тестируете некую функцию продукта. (С таким же успехом вы можете проверять и переменную. Тестирование не ограничивается функциями.)

Возможно, эта функция проблемная. То есть она может не соответствовать важному показателю качества продукта. Чтобы определить, так ли это, рассмотрим проблемные факторы — то, что повышает вероятность сбоя функции.

Атрибуты качества

Если в функции отсутствует или нарушен хотя бы один из этих атрибутов, то, скорее всего, о ней следует составить баг-репорт:

- доступность;
- возможности;

- совместимость;
- параллелизм;
- соответствие стандартам;
- эффективность;
- возможность установки и удаления;
- локализуемость;
- ремонтопригодность;
- производительность;
- переносимость;
- возможность восстановления;
- надежность;
- масштабируемость;
- безопасность;
- возможность сопровождения;
- тестируемость;
- удобство использования.

Чтобы определить, является ли функция дефектной, спросите себя, как вы докажете, что в ней отсутствует или нарушен один из этих атрибутов.

Например, рассмотрим *удобство использования*. Как доказать, что тестируемая функция неудобна в использовании? Как будет выглядеть неудобство? С помощью каких традиционных тестов на удобство использования можно изучить эту функцию? Задавайте подобные вопросы (и проводите соответствующие тесты).

Эти тесты ограничены вашим воображением, но многие из идей, которые могут прийти вам в голову, можно взять из списка проблемных факторов.

Проблемные факторы

Ниже приведены некоторые факторы, из-за которых могут возникать ошибки. Вы можете рассматривать каждый из них как маленький или большой (на ваше усмотрение) предупреждающий флажок и разработать тесты, позволяющие определить, действительно ли программа имеет уязвимость, на которую указывают эти факторы.

- **Новинки.** Новые функции могут не работать.
- **Новая технология.** Новые концепции приводят к новым ошибкам.

- **Новый рынок (рынки).** Участники иной клиентской базы будут по-другому воспринимать и использовать продукт.
- **Кривая обучения.** Ошибки совершаются по незнанию.
- **Изменения.** Могут привести к поломке старого кода.
- **Поздние изменения.** Ошибки могут возникать из-за поспешных решений, торопливых действий или деморализованности персонала.
- **Спешка в работе.** Некоторые задачи или проекты испытывают хронический недостаток финансирования, в результате чего страдают все аспекты качества работы.
- **Плохой дизайн или неподдерживаемая реализация.** Некоторые внутренние проектные решения настолько усложняют сопровождение кода, что исправления постоянно вызывают новые проблемы.
- **Уставшие программисты.** Длительная сверхурочная работа в течение нескольких недель или месяцев чревата неэффективностью и ошибками.
- **Другие проблемы персонала.** Проблемы с алкоголем, со здоровьем, смерть в семье, два программиста, которые не хотят взаимодействовать (как и их код), и т. п.
- **«Просто добавлю».** Любимая (но незапланированная) функция программиста может плохо взаимодействовать с другим кодом.
- **N.I.H.** (Not Invented Here, отказ от продукции, которая не произведена компанией). Внешние компоненты могут вызывать проблемы.
- **N.I.B.** (Not In Budget). Задачи, не предусмотренные бюджетом, могут быть выполнены некачественно.
- **Двусмысленность.** Неоднозначные описания (в спецификациях или других документах) могут привести к некорректным или противоречивым реализациям.
- **Противоречивые требования.** За двусмысленностью часто скрывается конфликт, результатом которого является потеря ценности продукта для какого-либо человека.
- **Неизвестные требования.** Требования возникают на протяжении всего процесса разработки. Невыполнение законного требования является нарушением качества для данного стейкхолдера.
- **Меняющиеся требования.** Люди понимают, чего они хотят, по мере разработки продукта. Придерживаясь списка требований, составленного в начале проекта, можно выполнить контракт, но не получить продукт. (См. www.agilealliance.org.)
- **Сложность кода.** Сложный код может содержать баги.

- **Наполненность багами.** Функции с большим количеством известных багов могут также содержать много неизвестных.
- **Зависимости.** Одни отказы могут вызвать другие.
- **Сложность тестирования.** Риск медленного и неэффективного тестирования.
- **Малый объем модульного тестирования.** Программисты сами находят и исправляют большинство своих ошибок. Сокращение времени чревато возникновением риска.
- **Малый объем тестирования системы.** Непроверенное программное обеспечение может привести к отказу.
- **Опора на привычные, но ограниченные стратегии тестирования.** Например, регрессионные и функциональные тесты могут привести к накоплению ошибок, сохраняющихся в разных версиях.
- **Слабые средства тестирования.** Если не существует инструментов, позволяющих выявлять и изолировать класс ошибок (например, указатели на несуществующий объект), то ошибка, скорее всего, останется незамеченной.
- **Неисправимость.** Риск невозможности исправить баг.
- **Ошибки, типичные для языка.** Например, указатели на несуществующий объект в языке C. См., например, книги *Pitfalls of Object-Oriented Development* (Webster, 1995) и *Java Pitfalls: Time-Saving Solutions and Workarounds to Improve Programs* (Daconta et al., 2000).

Использование списков ошибок

В книге *Testing Computer Software* (Kaner et al., 1993) приводится список из 480 распространенных дефектов. Вы можете использовать его или создать свой. Работать с таким списком можно следующим образом.

1. Найдите в списке дефект.
2. Спросите, может ли тестируемое программное обеспечение иметь этот дефект.
3. Если теоретически возможно, что в программе может быть дефект, то спросите, как можно найти баг, если он там есть.
4. Спросите, насколько правдоподобно, что этот баг может быть в программе, и насколько серьезным будет отказ, если он там есть.
5. При необходимости разработайте тест или серию тестов для выявления багов такого типа.

К сожалению, слишком многие тестировщики работают лишь со списком багов, представленным в книге *Testing Computer Software*. Он устарел. Он был устаревшим в момент своей публикации. И он не охватывает все проблемы *вашей* системы. Составление списка багов — непрерывный процесс, на который приходится постоянно тратить силы и время. Вот пример из практики Хунга Нгуена (Hung Nguyen) (личное сообщение):

> «Эта проблема возникла в системе "клиент/сервер". Система отправляет клиенту список имен, по которому проверяется уникальность имени, вводимого клиентом.
>
> Клиенты 1 и 2 одновременно вводят одно и то же новое имя. Оба экземпляра имени являются новыми по отношению к локальному списку сравнения и поэтому принимаются. Теперь у нас два экземпляра одного и того же имени.
>
> По мере обнаружения проблем мы добавляем их в библиотеку. Метод обнаружения является исследовательским и требует освоения базовой технологии. Выигрышные темы для тестирования фиксируются в таблицах или сценариях, которым предстоит автоматизация.
>
> По мере выявления новых проблем (в рамках ваших временны́х ограничений) добавляйте их в свою базу данных, чтобы тестировщики имели к ним доступ в следующем проекте».

Использование новейших источников для пополнения списков ошибок

Существует множество источников, где можно проверить распространенные отказы на общих платформах. Например, информацию, которую мы считаем полезной, можно найти здесь:

- www.cnet.com;
- книга Нгуена (Nguyen, 2001);
- книга Теллеса и Се (Telles, Hsieh, 2001).

Глава 4

Защита багов

Тестировщик, который не может составить хороший баг-репорт, похож на холодильник, в котором свет горит, только когда дверца *закрыта*. Ситуация может быть освещена очень хорошо — но не настолько, чтобы это имело значение. Вы информационная служба, но для того, чтобы быть эффективными, вам нужно делать нечто большее, чем просто заполнять шаблоны отчетов и предполагать, что они полностью понятны. Научитесь писать отчеты и представлять итоги тестов так, чтобы это *приносило* результат. Мы называем это защитой багов.

Вы — то, что вы пишете

Баг-репорты (bug reports, «отчеты о багах») — основной результат работы большинства тестировщиков. Благодаря этим документам читатели формируют представление о вас. Чем лучше отчеты, тем выше ваша репутация.

Программисты черпают из ваших отчетов важную информацию. Хороший отчет улучшает вашу репутацию. Плохой — порождает дополнительную (и, по мнению программистов, ненужную) работу. Если вы тратите много их времени, то они будут избегать вас и работы с вами.

Программисты не единственная ваша аудитория. Баг-репорты иногда читают руководители. Внутрикадровые проблемы быстро привлекают их внимание и раздражают их. Отчеты, которые выглядят обвиняющими, мелочными по тону, плохо объясняющими проблему, содержащими недостаточно исследований или наводящими на мысли о том, что вы раздуваете из мухи слона, создадут негативное впечатление у людей, от которых зависит ваше повышение по службе.

Все в выигрыше, если вы грамотно проводите тестирование и хорошо пишете ответы.

Защита багов способствует их исправлению

Каждый баг-репорт, который вы написали, — это документ, призывающий к исправлению бага.

Некоторые баги никогда не будут исправлены. Ваша ответственность не в том, чтобы убедиться, что все баги будут исправлены. Ваша задача — сообщать о багах таким образом, чтобы тот, кто будет читать отчет, мог понять все последствия проблемы.

Чем лучше вы проведете тестирование и составите отчет, тем выше вероятность того, что баг будет исправлен.

Сделайте ваш баг-репорт эффективным инструментом продаж

Независимо от того, думаете вы об этом таким образом или нет, ваш баг-репорт — это инструмент продаж; он создан для того, чтобы убедить людей расстаться с драгоценными ресурсами ради получения выгоды, которую вы предлагаете. Применительно к багам ресурсы — это время и деньги, а выгода — улучшение качества продукта, возникающее в результате исправления конкретного бага.

Стратегии продаж, как правило, предполагают достижение двух целей.

- **Преподнести выгоду так, чтобы ваш потенциальный клиент захотел ее получить.** Ваш отчет должен содержать сведения, благодаря которым читатель точно поймет, почему баг нужно исправить. Например, вы можете объяснить, как этот баг мешает нормальному использованию продукта, какие данные он повреждает или как часто люди будут с ним сталкиваться. Вы можете сослаться на рецензию в журнале или иную публикацию, в которой жалуются на подобный баг у конкурентов. Вы можете привести статистику технической поддержки, показывающую, что такие баги, как этот, в других продуктах стоили больших денег. Вы можете показать, что программа прошла данный тест в предыдущем выпуске (в некоторых компаниях это ключевая проблема). Или можете сделать так, чтобы отчет привлек внимание тех, чьей работе мешает этот баг.

Во многих случаях (см. ниже) вы можете взять незначительный на первый взгляд баг и обнаружить более серьезные последствия, проведя дополнительное тестирование, вместо того чтобы сообщать о первой версии бага, который вы обнаружили.

- **Предвидеть возражения со ссылкой на незначительность бага и парировать их.** Многие баги игнорируются как слишком незначительные, невоспроизводимые, непонятные; как те, которые вряд ли когда-либо возникнут в реальном мире; как проблемы, возникающие только при очень специфической конфигурации оборудования, которой ни у кого не будет; как баги, которые слишком рискованно исправлять; или как баги, которые вряд ли побеспокоят реальных пользователей продукта. Вы можете регулярно предупреждать возражения со ссылкой на незначительность выявленных багов, следуя рекомендациям по составлению баг-репортов. Пишите ясно и просто, проверяйте, повторяется ли баг в нескольких конфигурациях; если да, то отразите это в отчете. Другие возражения варьируются в зависимости от бага. Вы можете предвидеть их и включить соответствующую информацию в отчет или подождать, оценить первоначальную реакцию на отчет и добавить информацию по мере проведения баг-ревью. Мы *не говорим*, что в отчете вы должны доказывать серьезность бага. Старайтесь не говорить: «Я знаю, что вы не собираетесь исправлять этот баг, считая его неважным по *этой* причине, но вам следует знать *об этом*». Вместо этого мы предлагаем вот что: если вы в корне не согласны с возражением, то добавьте в баг-репорт некоторые факты, имеющие отношение к делу, например: «Аналогичный баг имелся во второй версии этого продукта. По оценкам руководителя службы технической поддержки, вследствие этого затраты на техподдержку превысили 100 000 долларов».

Ваш отчет об ошибке — это ваш представитель

Обычно вы не присутствуете при получении и прочтении вашего баг-репорта. Когда вы его пишете, то просите программиста (который не является вашим подчиненным) выполнить дополнительную работу. Программистам редко хватает времени для исправления всех багов. Бóльшую часть этой работы они выполняют в свое личное время — после окончания рабочего дня или в выходные дни. Вы просите их уделить это время найденному вами багу.

В некоторых компаниях (особенно по мере приближения к завершению проекта) различные руководители решают, что нужно исправить. Груп-

па лиц, принимающих решения, может называться советом по контролю изменений (в разных компаниях эта группа называется проектной командой, группой рассмотрения, боевой командой или просто командой баг-ревью). Эти люди понимают, что каждое изменение стоит денег, требует времени и несет в себе риск прекращения работы каких-либо иных компонентов.

Чтобы добиться исправления бага, вам необходимо убедить совет по контролю изменений одобрить исправление или убедить программиста исправить баг самостоятельно (возможно, поздно ночью, когда совет не видит[1]). Баг-репорт — это ваш основной (часто единственный) инструмент убеждения людей в необходимости исправления бага. Вы можете иметь возможность отстаивать свои интересы на заседании совета по контролю изменений, но во многих компаниях на нем присутствует только один тестировщик (возможно, ваш ведущий тестировщик или менеджер по тестированию). То, насколько хорошо этот человек будет защищать ваш баг, будет зависеть от вашего отчета.

Потратьте время на то, чтобы сделать ваши баг-репорты ценными

Баг-репорты читает и использует большое количество людей, поэтому уделите время тому, чтобы сделать каждый отчет информативным и понятным. Это повысит ценность вашей компании.

В большинстве компаний баг-репорты служат нескольким целям. Например:

- предупреждают о дефектах и помогают программистам устранять основные проблемы;
- предупреждают людей о проблемах в спецификациях и (в зависимости от политики компании), возможно, в тестовой документации, документации пользователя или средствах разработки;
- предоставляют справочную информацию для технических писателей, разрабатывающих разделы по устранению неисправностей для руководства пользователя или сайта компании;

[1] Привлечение людей к улучшению продукта за спиной совета по контролю изменений — отличная стратегия в одних компаниях и абсолютно неправильная в других. Обратите внимание на свою корпоративную культуру.

- отмечают вопросы, над которыми, возможно, стоит поработать в процессе обучения клиентов;
- предоставляют наиболее важную информацию для послепродажной поддержки клиентов, столкнувшихся с проблемами, которые не были устранены вовсе или устранены не полностью;
- предоставляют руководству информацию о состоянии и качестве разрабатываемого продукта;
- представляют стартовые предложения по улучшению продукта на начало его следующего выпуска.

Любой стейкхолдер должен иметь возможность сообщить о баге

Стейкхолдер (stakeholder) — человек, лично заинтересованный в успехе продукта. Это может быть сотрудник компании, производящей продукт, клиент или пользователь, который в значительной степени зависит от продукта.

Вся ваша компания заинтересована в выпуске высококачественного продукта в рамках бюджета, в срок и с нужными функциями. Любой человек, работающий в компании, убежденный в том, что качество или набор функций не годятся, должен иметь возможность выразить свой протест таким образом, чтобы он дошел до команды разработчиков. Команда тестировщиков никогда не должна препятствовать такому общению, а, наоборот, всегда должна способствовать ему, даже если (особенно если) жалобы поступают не от разработчиков продукта (в том числе от участников ее самой).

Вы можете не показывать записи отслеживания багов всем желающим. Такие данные должны рассматриваться как конфиденциальные, если вы считаете исходный код и другие решения по разработке конфиденциальными. Тем не менее широкий круг людей должен иметь возможность получить информацию *из* ваших записей.

Будьте осторожны, меняя формулировки в баг-репортах других людей

Добавляйте комментарии, но не редактируйте чужие отчеты без разрешения тех, кто их написал. Вы бы не хотели, чтобы ваше имя стояло под чем-то, чего вы не писали и не одобряли. Того же не хотят и другие

люди в вашей компании, которые сообщают о багах, даже те, кто пишет отвратительные баг-репорты.

Изменение формулировок в отчете без разрешения также может привести к потере важной информации.

Всякий раз, когда вы добавляете что-либо в баг-репорт, особенно в чужой, вставляйте комментарий с вашими инициалами и датой. Например, обычно дополнительная информация выносится на новую строку отчета и начинается с фразы наподобие «[СК 12/27/01]». Это также позволит людям задавать уточняющие вопросы нужному человеку.

Сообщайте о замеченных недостатках качества как о багах

«*Для кого-то качество — это ценность*». (Weinberg, 1992).

У разных людей разные ожидания от продукта, но если какой-то стейкхолдер им разочарован и считает, что тот стал менее ценным из-за чего-то, что он делает или не делает, то должен написать об этом в виде баг-репорта.

Ваша задача как тестировщика — помочь этому человеку составить отчет, в котором его опасения будут изложены четко и убедительно.

Некоторые стейкхолдеры не могут сообщать о багах. Вы — их доверенное лицо

Заказчики коммерческого готового программного обеспечения (например, потребители) не могут сообщать о багах во время разработки, поскольку у них еще нет программы. Другие стейкхолдеры проекта могут быть недоступны. Как тестировщик, вы заменяете отсутствующие заинтересованные стороны. Чтобы понять, о чем сообщать от их имени, и сделать ваши отчеты убедительными, вам необходимо изучить, как эти люди будут использовать продукт и что будут ценить или ненавидеть в продукте такого типа.

Привлеките внимание стейкхолдера к спорным багам

Допустим, вы считаете, что убедить программистов исправить баг будет трудно, но хотите, чтобы это было сделано. В таком случае подумайте,

кому еще в компании выгодно, чтобы баг был исправлен. Например, несоответствия в пользовательском интерфейсе программистам могут показаться незначительными. Попытки переубедить руководителей проекта в подобных вопросах — пустая трата времени.

Несоответствия приводят к увеличению затрат на документацию, обучение и техническую поддержку. Кроме того, если они неизбежно проявятся при демонстрации продукта, это может привести к снижению продаж. Если несоответствия влияют на доступность для лиц с ограниченными возможностями, то государственные учреждения, требующие соблюдения норм доступности, могут отказаться от приобретения продукта. Если несоответствия будут очевидны для рецензентов, то это может привести к негативным отзывам в прессе (описанию продукта как небрежно спроектированного или сложного в использовании). Ни один из этих видов затрат не ударил по бюджету программистов.

Напишите баг-репорт (или служебную записку, приложенную к его копии) так, чтобы он привлек внимание людей, бюджет которых пострадает из-за этого бага. Они сами будут доказывать вам, что баг необходимо исправить. Но если они посчитают баг несерьезным, то не будут настаивать на его исправлении. Тогда, возможно, вам следует оставить этот баг в покое и начать выступать за исправление какого-либо другого бага.

Никогда не используйте систему отслеживания багов для контроля работы программистов

Велик соблазн сообщить, что у одного программиста найдено огромное количество багов, что он их постоянно исправляет или пытается все отложить. Однако после того, как вы используете данные системы отслеживания для того, чтобы враждебно покритиковать одного программиста или поставить его в неловкое положение, остальные программисты займут оборонительную позицию относительно этой системы. Как следствие, наиболее вероятно, они будут утверждать, что баги в дизайне — это не баги, что похожие баги — это дубликаты, что о невоспроизводимых багах не следует сообщать и что вы некомпетентны или нечестны в своем тестировании. Это вполне разумно. Как только вы (или кто-либо другой) начнете использовать систему отслеживания багов не в технических, а в политических целях или для управления персоналом, люди будут относиться к ней именно так.

Никогда не используйте систему отслеживания багов при оценке эффективности тестировщика

Если вы (руководитель отдела тестирования) будете поощрять своих подопечных за количество найденных ими багов, вы вряд ли останетесь довольны их работой. Чтобы набрать нужное количество отчетов об ошибках, тестировщики могут начать, к примеру, сообщать о незначительных багах, которые легко найти, или же создавать отдельные отчеты для каждого частного случая одной и той же ошибки. Из-за этого они, скорее всего, перестанут выделять время на обучение других тестировщиков и улучшение системы отслеживания багов (или любой другой). А разработчики, в свою очередь, начнут упускать ошибки проектирования, теряющиеся в множестве отчетов, созданных для количества.

Сообщайте об ошибках своевременно

Не откладывайте написание отчета об ошибке на завтра: вы можете забыть о значимых деталях. Чем дольше вы откладываете, тем меньше вероятность, что ошибка будет исправлена.

Более того, если руководители знают о том, какую часть кода вы тестируете, и при этом не видят сообщений об ошибках в ней, они могут подумать, что код исправен и работает стабильно.

Никогда не рассчитывайте на то, что очевидный баг уже задокументирован

Другие могут подумать так же, и в результате некоторые очевидные баги будут зафиксированы только на этапе бета-тестирования. Если вы думаете, что отчет об ошибке в системе отслеживания уже есть, найдите его и посмотрите, как именно ошибка была описана и каков был отклик. Возможно, ваш дополнительный комментарий будет полезен. А возможно, баг-репорт придется переписать, если он изначально был плохо составлен или же отклонен.

Мы видели, как это происходило с некоторыми серьезными багами. Многие люди знали об этом баге, но считали, что о нем сообщил кто-то другой. В одном случае баг был исправлен только после того, как продукт был выпущен.

Сообщайте об ошибках проектирования

Программа — это часть системы, включающей в себя материальное оборудование, другие программы и людей, для удобства которых она создается. Если программа сложна в использовании, конфликтует с другим ПО или требует специального аппаратного обеспечения, ее ценность становится сомнительной. Может быть, вы единственный член команды разработчиков, который способен заметить ошибку в проектировании и оценить ее влияние на действующую систему в целом. И если не вы сообщите о такой проблеме, то кто это сделает?

Некоторые (в том числе консультирующие специалисты) считают, что тестировщики не должны сообщать об ошибках проектирования, поскольку, во-первых, архитектура программы должна утверждаться на ранних этапах разработки, а во-вторых, тестировщики не обладают необходимыми знаниями в этой области. Мы не согласны ни с самим утверждением, ни с аргументами, его подкрепляющими.

- **Поздняя критика проекта.** Даже если программа была полностью спроектирована заранее (что вообще редко происходит в реальных проектах в реальном мире), невозможно просчитать наперед все ошибки, связанные с ее применением. Вполне нормально, что погрешности проектирования могут проявиться тогда, когда система будет уже введена в эксплуатацию[1]. Ошибки нужно документировать по мере их обнаружения, а выявить некоторые из них возможно только на финальных этапах работы над продуктом.
- **Недостаточная экспертность тестировщиков.** С одной стороны, не все тестировщики способны адекватно оценивать проектные решения. С другой стороны, тестировщики — одни из немногих членов команды, которые досконально проверяют работу готовой программы в целом перед ее продажей или запуском в производство. Будьте, однако, осторожны с критикой, если у вас недостаточно опыта в определенной области. Например, прежде чем критиковать пользовательский интерфейс, сначала стоит

[1] Этим объясняется рост интереса к таким современным подходам к разработке, как экстремальное программирование (Extreme Programming) и рациональный унифицированный процесс (Rational Unified Process) (в общем, подходы, называющиеся итеративными, адаптивными, эволюционными или гибкими). Эти подходы основываются на том, что требования к продукту могут меняться даже на финальных стадиях разработки, а их использование призвано минимизировать риски, вызываемые этими изменениями. См., например, Beck, 1999, Kruchten, 2000 и www.agilealliance.org.

ознакомиться с руководством по его дизайну, разработанным для вашей системы[1], и посоветоваться с теми, кто знает о дизайне больше, чем вы. Если же вы уверены в ошибочности того или иного решения, то смело регистрируйте ошибку в системе отслеживания.

Чтобы команда тестировщиков была более эффективной в обнаружении ошибок проектирования, она должна состоять из людей, имеющих за плечами разнообразный опыт: например, человек с опытом работы в конкретной предметной области (то есть обладающий глубоким знанием о том, как и почему возникла необходимость в разработке данного продукта) будет обращать повышенное внимание на проблемы, которые могут затронуть конечного пользователя, — и это только одна сторона вопроса. Если один из тестировщиков раньше занимался проектированием баз данных, другой — сетевой безопасностью, третий — пользовательскими интерфейсами и т. д., то команда в целом будет способна грамотно оценивать различные аспекты проектирования всего продукта[2].

Ошибки переполнения могут привести к уязвимостям в системе безопасности

Большинство случаев взлома систем через Интернет происходит из-за ошибок, связанных с переполнением буфера (Schneier, 2000a, 2000b). Их можно было бы отлавливать до релиза, если бы тестировщики усиленно не закрывали глаза на граничные случаи, до которых программистам, по их мнению, нет дела. Все, что может повредить данные или помешать программе нормально работать, — потенциальная лазейка в системе безопасности.

Переполнение буфера происходит тогда, когда размер вводимых данных превосходит отведенное под них место. Часть данных, которая выходит за границы, помещается в другие области в памяти. Дальнейшее развитие событий зависит от характера данных и от того, как определенное место в памяти предполагалось использовать. Иногда не

---

[1] Например, Apple, Microsoft, Sun и несколько других компаний опубликовали руководства по дизайну графических пользовательских интерфейсов.

[2] Не всегда возможно в сжатые сроки сформировать команду, состоящую из тестировщиков с необходимым (для конкретного проекта) опытом. Возможно, вам придется нанимать специалистов со стороны (подрядчиков или консультантов), чтобы они обучили вашу команду или провели какую-либо оценку.

происходит ничего, иногда на экране появляются помехи, иногда программа становится неуправляемой, а иногда и вовсе ломается. Умелый взломщик может воспользоваться такими лазейками, чтобы проникнуть в систему, на которой запущена программа, и получить контроль либо над ней, либо над системой в целом.

Предположим, можно сломать программу, введя 65 536 девяток в поле, принимающее значения от 1 до 99. Будет ли кто-то так делать в своем уме? Разумеется, будет. Наверное, не ваши друзья и не программисты, которые не будут утруждать себя исправлением этой ошибки, ссылаясь на то, что «если какой-то идиот сотворил такое, то сам виноват». Но подобные действия будут совершать не только идиоты.

Любой баг, приводящий к серьезным последствиям, должен быть устранен вне зависимости от того, насколько «маловероятно» его появление. Обнаружив уязвимость, хакер может написать вредоносный код для атаки на систему, а этим кодом смогут потом воспользоваться менее опытные хакеры-дилетанты.

Не игнорируйте крайние случаи

Эффективное тестирование часто включает в себя проверки на предельных значениях, ведь именно при их обработке обнаруживаются первые ошибки. Логика проста: если программа адекватно обработала такие значения, значит, скорее всего, она будет стабильно работать и в других случаях. Получается, о качестве программы можно судить по тому, как она реагирует на предельные значения. Например, при проверке поля ввода, которое должно принимать значения от 0 до 999, разумнее всего будет подать на вход 1, 0, 999 и 1000.

Некоторые программисты закатывают глаза, сталкиваясь с тестами на одном или нескольких предельных значениях, называя их «патологическими» случаями. Вероятность того, что подобный случай встретится на практике, стремится, по их мнению, к нулю, поэтому они считают, что возникающие при этом баги можно не исправлять.

Представим, что программа, которая должна принимать входные значения от 0 до 999, неадекватно срабатывает при вводе 999. Если вы остановитесь на этом значении, программисты, скорее всего, проигнорируют это сообщение об ошибке: «Кому взбредет в голову вводить 999? Это же крайний случай. В реальности никто так делать не будет!»

Попробуйте ввести другие значения. Вводите, к примеру, числа поменьше до тех пор, пока не установите, на каком промежутке программа работает нормально. Если она не справляется со значениями больше 99 (то есть с достаточно заметным промежутком от 100 до 999), такой отчет программистам будет трудно проигнорировать. Если же в отчете об ошибке говорится только о крайнем случае, это тоже значимо — только сообщите об этом явно.

Как и с последующим тестированием, время, которое вы можете потратить на такие ошибки, ограничено. Если вы понимаете, что работа над проблемой затянулась, переходите к поиску других багов. Если ваши коллеги-программисты воспринимают тесты на предельных значениях всерьез, им будет достаточно одного такого теста.

Незначительные ошибки — тоже ошибки

Несущественные ошибки могут смутить пользователей и испортить их впечатление о продукте в целом. Незначительными багами называют орфографические ошибки; ошибки форматирования изображений; пиксельные пятна, остающиеся после перемещения курсора; неточности в вычислениях; графики, не соответствующие масштабу; ошибки в справке; выделенные серым опции в меню, которые на самом деле активны (и кликабельные опции, которые отключены); горячие клавиши, которые не срабатывают; некорректные сообщения об ошибках; неправильную обработку предельных значений; неправильную обработку тайм-аутов; и вообще все, что программист посчитает не стоящим потраченных усилий.

Несколько лет назад Кейнер, один из авторов этой книги, совместно с Дэвидом Пелсом изучал зависимость между затратами на техническую поддержку и незначительными багами. Кейнер был руководителем проектов по разработке; Пелс был руководителем службы технической поддержки. Работая над хорошо продающимся продуктом, они досконально изучали все: письма клиентов, рецензии в журналах, записи телефонных разговоров с пользователями, отчеты об ошибках в багтрекере. Они называли «дешевыми» проблемы, на устранение которых требовалось меньше четырех часов (не считая времени, потраченного на их обнаружение): решение таких проблем требовало либо изменений в коде, либо замены драйвера или файла с данными на диске, либо внесения поправок в руководство пользователя или в текст на упаковке продукта. Если работа над решением проблемы занимала меньше

четырех часов и предотвращала дальнейшие жалобы пользователей, то эта проблема считалась дешевой. Решение попадающих под этот критерий дешевых проблем (или незначительных багов) могло бы не допустить больше половины обращений в техническую поддержку. (Эта история кратко описывается в их совместной книге, см. Kaner, Pels, 1997.) В течение года команда Кейнера устраняла многие из таких дешевых ошибок, и в итоге им удалось сократить затраты на техническую поддержку вдвое. И хотя здесь нельзя говорить о прямой причинно-следственной связи между одним и другим, поскольку команда Кейнера также чинила более серьезные поломки и вводила новые компоненты в свой продукт (вместе с которыми возникали и новые баги), Кейнер и Пелс пришли к очевидному выводу, что такая общая «чистка» в значительной степени повысила удовлетворенность пользователей. Доля продукта на рынке за этот год также сильно увеличилась.

Наличие нескольких незначительных дефектов допустимо в любом продукте. Но чем их больше, тем быстрее обманывается доверие клиента. Но еще более губительны последствия игнорирования таких багов. Если вы будете игнорировать мелкие баги (а некоторые компании заставляют своих сотрудников сообщать только о «стоящих» ошибках), со временем вы начнете закрывать глаза и на более серьезные. В конечном счете продукт на рынке провалится (если вы, конечно, не монополист в своем сегменте). Пример такой терпимости к крупным ошибкам, ярко описанный Ричардом Фейнманом в книге «Не все ли равно, что думают другие?», — катастрофа шаттла «Челленджер» (Feynman, 1989).

Различайте серьезность и приоритет

Серьезность относится к степени влияния ошибки на работу программы и последствиям этой ошибки. *Приоритет* обозначает сроки, в которые задача должна быть решена. Серьезность остается неизменной, если только вы не обнаружите скрытые последствия ошибки. Приоритеты же могут меняться по мере готовности проекта. Вообще говоря, у серьезных ошибок высокий приоритет, а у мелких и косметических ошибок приоритет низок. Тем не менее серьезность и приоритет — не одно и то же. Некоторые серьезные ошибки не стоят того, чтобы тратить время и усилия на их исправление: например, вы нашли баг, искажающий любую новую запись, созданную до декабря 1999 года. Сколько-то лет назад (когда декабрь 1999-го еще был впереди) у этой ошибки был бы высокий приоритет. Теперь, хотя эта ошибка и остается серьезной, многие компании не будут ее исправлять.

С другой стороны, если на заставке, появляющейся при запуске программы, логотип вашей компании перевернут, а ее название написано с ошибкой, то это считается исключительно косметической проблемой — однако во многих компаниях ей будет присвоен высокий приоритет.

Отказ — это симптом ошибки, а не она сама

Когда вы видите отказ, вы смотрите на неправильное поведение программы, а не на ошибку в коде. Отказ может выглядеть незначительным (например, следы на экране после перемещения указателя мыши), но основная проблема может быть гораздо более серьезной (например, указатель на несуществующий объект). Если вы столкнетесь с тем же самым плохим кодом в несколько иных условиях, то увиденное вами неправильное поведение может быть еще более некорректным.

Поэтому всякий раз, когда вы видите ошибку, которая выглядит незначительной, проведите дополнительное тестирование, чтобы:

- выявить более серьезные симптомы, — это позволит составить более убедительный отчет;
- показать, что проблема возникает в более широком диапазоне обстоятельств и ее заметит множество людей.

Если проблему трудно воспроизвести:

- проведите дополнительное тестирование, чтобы определить критические условия, при которых проблема станет воспроизводимой, а затем проведите еще одно тестирование, чтобы воспроизводимая ошибка теперь выглядела неприятно.

Проведите дополнительное тестирование, казалось бы, незначительных ошибок кода

Столкнувшись с ошибкой в коде, программа оказывается в состоянии, которого программист не предполагал и, возможно, не ожидал. В данных могут оказаться недопустимые значения. Если продолжать использовать эту программу после первого отказа, то может произойти все что угодно.

Встретив незначительный отказ, не ограничивайтесь простым составлением баг-репорта. Программа находится в уязвимом состоянии. Попробуйте этим воспользоваться. Продолжайте тестировать ее — и,

возможно, обнаружите, что реальные последствия ошибки гораздо серьезнее, например сбой системы или повреждение данных.

Мы рекомендуем попробовать три вида дополнительного тестирования.

- **Меняйте свое поведение** (варьируйте условия, меняя свои действия). Например, вызывайте отказ снова и снова. Есть ли накопительный эффект? Попробуйте выполнить действия, *связанные с задачей*, которая завершилась неудачно. Например, если программа неожиданно, но слегка прокручивает дисплей при сложении двух чисел, то попробуйте тесты, которые влияют на сложение или на числа. Попробуйте выполнить тесты, *связанные с отказом*. Если его причиной является неожиданная прокрутка дисплея после сложения, то попробуйте сначала прокрутить, а затем выполнить сложение. Попробуйте перерисовать экран, а затем выполнить сложение. Попробуйте изменить размер отображения цифр, а затем выполнить сложение. Или попробуйте вводить цифры быстрее, или *измените скорость* своих действий каким-либо другим способом. И конечно, можно попробовать множество других техник исследовательского тестирования. Иногда полезно провести большое количество совершенно не связанных между собой дополнительных тестов, просто продолжая использовать программу, не перезапуская и не перезагружая ее.
- **Меняйте параметры и настройки программы** (варьируйте условия, меняя что-либо в тестируемой программе). Типичные изменения — использование другой базы данных, изменение значений постоянных переменных, способа использования памяти программой или всего остального, что она позволяет изменять (любой другой параметр, предпочтение или настройка). В примере с прокруткой можно изменить размер окна программы, точность отображаемых цифр или настройку фоновой активности для проверки орфографии.
- **Меняйте программную и аппаратную среду** (варьируйте условия, меняя ПО, работающее с этой программой, или оборудование, на котором вы ее тестируете). Это не тестирование конфигурации. Вы не проверяете баг на стандартных конфигурациях. Вместо этого вы задаетесь вопросом, как можно изменить конфигурацию, чтобы сделать *этот отказ* более впечатляющим. Например, если вы считаете, что отказ может быть вызван проблемами с синхронизацией, то используйте другой процессор, видеодисплей или коммуникационное оборудование. Если считаете, что важным фактором может быть память, то поработайте на машине с меньшим (или большим) объемом памяти или измените настройки виртуальной памяти.

Дополнительное тестирование можно проводить бесконечно. Но как много тестов вы должны делать? Уделяйте хотя бы несколько минут тестированию каждого отказа, который, по вашему мнению, отображает ошибку кода (нечто непреднамеренное со стороны программиста).

При некоторых отказах вам придется провести дополнительное тестирование, возможно, в течение целого дня. Доверьтесь своей интуиции. Если вы считаете, что еще несколько тестов могут дать новую ценную информацию, то попробуйте их провести. Если считаете, что сейчас знаете о баге столько же, сколько узна́ете после дополнительного тестирования, то прекратите этот вид проверки и сообщите о том, что вам известно. Доверьтесь своей интуиции.

Всегда сообщайте о невоспроизводимых ошибках; они могут оказаться бомбами замедленного действия

Невоспроизводимые ошибки могут быть самыми дорогостоящими ошибками, которые может выпустить ваша компания. Иногда программа допускает ошибки, которые не удается воспроизвести. Вы видите отказ один раз, но не знаете, как получить его снова. Если с отказом столкнется клиент, то это подорвет его доверие к продукту. Если сотрудникам службы поддержки приходится прикладывать много усилий, чтобы оценить данные или среду клиента, это вызовет у него еще большее раздражение.

У программистов есть инструменты, которых нет у вас. Если вы четко изложите симптомы, то программист зачастую сможет проследить за кодом, спросив, как вы могли получить то или иное сообщение или что могло произойти, когда вы просматривали то или иное диалоговое окно или нажимали на тот или иной элемент управления. Небезосновательно считается, что программисты могут исправить 20 % «невоспроизводимых» ошибок, о которых им сообщают[1].

Даже если вы работаете в компании, программисты которой игнорируют невоспроизводимые баги, мы советуем вам все же сообщать о них.

---

[1] Очевидно, что в разных продуктах и наборах инструментов, доступных программистам, этот процент будет разным. Мы видели, как программисты работали с таким коэффициентом успешности в коммерческом ПО. Люди, заслуживающие доверия, рассказывали нам о программистах и проектах, в которых этот показатель достигал 80 %.

Не сообщайте о баге, который вы *не попытались* корректно воспроизвести. Но если вы все же не можете его воспроизвести, то о нем все равно стоит сообщить. Периодически сканируйте базу данных в поисках закономерностей среди невоспроизводимых багов. Одна и та же проблема может возникать несколько раз и (если вы всегда сообщаете о невоспроизводимых ошибках) будет фигурировать в разных отчетах. Каждый из них будет немного различаться. В совокупности они могут дать серьезные подсказки о том, что лежит в основе проблемы.

Сообщая о невоспроизводимом баге, уточните, что не можете его воспроизвести. В некоторых системах отслеживания есть соответствующее поле («Можете ли вы воспроизвести это: да/нет/периодически/неизвестно»). Некоторые компании вместо этого используют обозначения: например, помечают баг как NR (not reproducible, «невоспроизводимый») в итоговой строке или первой строке описания.

Использование клавиши PrintScreen, программы записи экрана типа Spector или даже видеозаписи может помочь вам доказать существование багов, которые, по мнению некоторых программистов, никогда не могли появиться.

Невоспроизводимые баги воспроизводимы

Отказы происходят при определенных условиях. Если вы их знаете, то можете воссоздать отказ. Если нет, то, скорее всего, у вас ничего не получится.

Если вы сталкиваетесь с багом, который не можете воспроизвести, значит, есть что-то важное, чего вы не замечаете. Если бы вы смогли выяснить, что это за условие, то смогли бы воссоздать баг. Ниже приведены несколько примеров условия, о которых мы сочли полезным упомянуть (то, на что люди часто не обращают должного внимания, разве что в ретроспективе).

- Возможно, вам встретилась трудноуловимая скрытая ошибка, например утечка памяти, указатель на несуществующий объект или поврежденный стек. Если вы подозреваете, что проблемы развиваются со временем, попробуйте совместно с программистами использовать Bounds Checker (www.numega.com), Purify (www.rational.com/products/purify_unix/index.jsp) или аналогичные инструменты либо прибегните к методам, позволяющим отслеживать поведение программы (записывайте вывод на видео, воспользуйтесь утилитой, которая делает снимок экрана каждую секунду, отслеживайте использование памяти и т. д.).

- Баг может проявиться только при первой установке или использовании продукта либо определенной его функции. Применяйте Drive Image (www.powerquest.com/driveimage/), Ghost (www.symantec.com/sabu/ghost) или аналогичный инструмент, позволяющий создать точную копию чистой системы (той, которая еще не видела ваше приложение). Восстановите чистую систему, перезагрузите приложение и посмотрите, сможете ли вы воспроизвести проблему теперь.
- Баг может зависеть от конкретных значений данных или от поврежденной базы данных.
- Баг может возникать только в определенное время или дату. Ищите ошибки, возникающие в выходные, в конце дня, квартала или года.
- Возможно, баг зависит от выполнения ряда взаимосвязанных задач в определенном порядке. Что вы делали перед тем, как выполнить задачу, за которой последовал сбой?
- Баг может быть следствием предыдущего отказа. Например, перезапускали ли вы машину после последней ошибки нарушения общей защиты (general protection fault, GPF)?
- Баг может возникнуть как результат взаимодействия тестируемого приложения с другим ПО, которое работает в фоновом режиме, или программами, конкурирующими с вашим приложением за доступ к устройству. Возможно, отказ свидетельствует о неожиданной проблеме при взаимодействии с устройством.

Существует и ряд других идей — их слишком много, чтобы можно было всесторонне описать в этой книге. Мы подробно рассматриваем эту область в первом томе *(Tester's Volume)* новой редакции книги *Testing Computer Software* (Kaner et al., 1999), а пока в поисках идей можете изучить книги Нгуена (Nguyen, 2000), Кейнера и др. (Kaner et al., 1993) и Теллеса и Се (Telles, Hsieh, 2001).

Учитывайте стоимость обработки ваших баг-репортов

Мы уже говорили вам о необходимости сообщать обо всех мелких и невоспроизводимых проблемах. Теперь нам придется немного отступить от этого правила и попросить вас проявить благоразумие в отношении того, что и как вы сообщаете.

Баг-репорты требуют затрат на обработку. Чтобы написать отчет, нужно время. Затем его читает программист и/или руководитель проекта.

В одних компаниях совет по контролю изменений проводит баг-ревью, чтобы решить, что делать с багом. В других компаниях баг попадает на рассмотрение совета по контролю изменений только после того, как программист или руководитель проекта принимает решение не исправлять его. Таким образом, читателей множество. Суммируйте время каждого — и обнаружите, что на анализ и отклонение мелкого бага в вашей компании уходит от двух до восьми человеко-часов. Если на поздних этапах проекта вы завалите компанию отчетами о мелких багах, то можете в одиночку снизить до нуля производительность команды программистов и руководителей проекта. Вы можете вызвать шквал неожиданного недовольства или гнева, сообщая об очень незначительных проблемах или составляя непонятные (отнимающие неоправданно много времени на прочтение) отчеты на поздних этапах проекта.

Когда вы сообщаете о мелком баге, позаботьтесь о том, чтобы ваш отчет был понятным и хорошо проанализированным (см. пункты о дополнительном тестировании далее в главе). Если вы собираетесь сообщить о куче мелких багов на поздних стадиях разработки, то обсудите со своим руководителем командную политику. Возможно, вам стоит отправлять баг-репорты во вторичную базу данных, которую официально рассмотрят в качестве исходного материала для следующего выпуска, но проигнорируют в текущем. Таким образом ваша работа сохранится и вы сможете переключиться на другие задачи.

Когда вы собираетесь сообщить о невоспроизводимом баге, особенно если делаете это на поздних стадиях разработки, постарайтесь воспроизвести его, действуя максимально осторожно. Если вы провели тщательное дополнительное тестирование и по-прежнему не можете воспроизвести баг, то укажите, что он невоспроизводим, и опишите, как вы устраняли неполадки и с какими симптомами столкнулись.

Особое внимание уделяйте багам, связанным с инструментами или средой

Если программа выходит из строя из-за известной уязвимости в ОС или в другом приложении или системе, с которой взаимодействует ваша программа, и этот отказ полностью вне контроля программистов, то вы можете принять обоснованное решение не сообщать о нем. С другой

стороны, вы могли обнаружить сбой, который легко устранить, добавив соответствующий код обработки ошибок.

Однако сбой в программе может быть вызван известной уязвимостью в ОС (или в каком-либо другом продукте или системе). В таких случаях у программиста может получиться переработать программу таким образом, чтобы она не выполняла критический системный вызов и, следовательно, не вызывала аварию в системе. Отчасти это баг ОС, но поскольку отказ можно предотвратить на уровне приложения, то это также баг приложения. Если вы считаете, что дело обстоит именно так, сообщите об этом следующим образом. Как обычно, сначала опишите действия по воспроизведению проблемы. Но затем добавьте примечание о том, что вы знаете, что это связано со взаимодействием между приложением и ОС, но надеетесь, что приложение сможет устранить баг на уровне ОС. Прежде чем сообщать о большом количестве багов таким образом, посоветуйтесь с человеком, которому вы доверяете и который хорошо разбирается в базовой системе.

Кроме того, рассмотрите возможность использования соответствующих инструментов для создания снимков конфигурации, в которой возникает отказ. Например, компания Rational распространяет бесплатную программу — анализатор установки, позволяющий записывать подробности установки Windows и сравнивать их с эталоном (VeriTest-Rational Install Analyzer, доступен по адресу www.rational.com/products/testfoundation). Подобным инструментом является InCtrl5, доступный по адресу zdnet.com.

Аналогично программисты часто возлагают вину за отказы на сторонний код, интегрированный в приложение. Сообщайте о таких багах в каждом случае, поскольку эти отказы вызваны ошибками в пакете кода, который ваша компания предоставляет клиентам. Выступайте за устранение этих багов, если имеете основания полагать, что программисты могут решить проблемы в коде сторонних разработчиков, который они используют.

Спросите, прежде чем сообщать о багах в прототипах или ранних частных версиях

Иногда программист показывает вам частную версию и просит провести закрытое тестирование. Если в вашей компании это разрешено, то соблюдайте негласные правила. Баги, обнаруженные при тестировании ранних версий кода, вероятно, не фиксируются в системе отслеживания. Вместо этого о них сообщается в разговоре, в заметках

или электронных письмах программисту. Если вы обнародуете ошибки, которые программист показал вам в ранней, закрытой версии кода, вы подорвете доверие программиста. Весьма вероятно, что вы больше не сможете увидеть эти ранние выпуски.

В определенный момент программа станет общедоступной в компании и будет готова к регулярному тестированию. Используйте заметки, которые вы делали во время закрытого предварительного тестирования программы. Все обнаруженные баги, которые еще не исправлены, должны попасть в систему отслеживания.

Нельзя не отметить злоупотребления этой схемой. Мы сталкивались с программистами, которые пытались скрыть свои проблемы с графиком, пряча свои баги. Они называли каждый выпуск программы для тестирования прототипом или частной версией вплоть до выпуска окончательной версии. Они не хотели, чтобы любые баг-репорты вносились в систему отслеживания (где их могло найти руководство) до тех пор, пока не появится финальная версия.

Если вы столкнулись с такой ситуацией, то обратитесь к своему руководству. Если вы руководитель команды тестировщиков, то, возможно, вам придется проконсультироваться с руководителем проекта или другими ключевыми стейкхолдерами. Основной вопрос заключается в том, считает ли ваша компания предварительные данные отслеживания багов инструментом, предназначенным в первую очередь для программистов, или корпоративным ресурсом, доступ к которому имеют ключевые стейкхолдеры. Постарайтесь быть готовыми к сотрудничеству и помочь. И ни в коем случае не участвуйте в действиях, которые ваша компания расценит как сокрытие информации.

Дублирующиеся баг-репорты — это самоисправляющаяся проблема

Конечно, обычно лучше добавить данные в существующий открытый баг-репорт, чем составлять новый о том, что, по вашему мнению, является тем же самым багом. Если вы знаете, что сообщаете о баге, который похож на другой, сделайте перекрестные ссылки на него.

В некоторых компаниях, особенно в тех, где уделяется слишком много внимания статистике багов, руководители проектов и программисты расстраиваются из-за дублирующихся баг-репортов или утверждают в разговоре с руководством, что количество багов раздуто за счет ду-

бликатов. В таких компаниях следует провести беглый поиск предыдущего отчета об этом же баге. Однако имейте в виду следующие соображения.

- Не позволяйте затягивать время поиска. На поиск дубликатов можно напрасно потратить слишком много времени, если база данных достаточно велика. Управляйте своим временем.
- Каждый хорошо написанный отчет об одном и том же баге содержит новую информацию, которая может помочь облегчить его устранение.
- Вопрос о том, являются ли два отчета дубликатами, может быть спорным. Возможно, два отказа вызваны одним и тем же сбоем. Или же в одном сбое могут быть несколько отказов. Пока вопрос не решен и баг не исправлен, сохраняйте всю собранную информацию (Pettichord, 2000a).

Руководитель проекта, заметивший, что дубликаты приводят к увеличению количества багов, может легко решить эту проблему, добившись их исправления.

Каждый баг заслуживает отдельного отчета

Не объединяйте разные баги в один отчет, пытаясь успокоить руководителя проекта или программиста, который постоянно жалуется на дублирование. Если вы напишете отчет, в котором будет указано множество ошибок, некоторые из них, скорее всего, никогда не будут исправлены. Когда вы сравниваете баг в базе данных с багом, о котором хотите сообщить, используйте вот какой критерий: если для проверки того, исправлен ли ваш баг и баг в базе данных, вы будете проводить разные тесты, то сообщайте о них как о разных багах.

Петтикорд рекомендует справляться с подобными багами немного подругому:

> «Иногда в интересах эффективности можно добавить в отчет список схожих мелких багов, предполагая, что все они могут быть исправлены сразу. Если какие-либо из них остаются открытыми после того, как отчет помечен как исправленный, то создайте новые отчеты по каждому из открытых багов со ссылкой на старый, который теперь можно пометить как исправленный».

Строка резюме — самая важная в баг-репорте

Это поле, иногда называемое заключением, имеет решающее значение, поскольку руководитель проекта и другие начальники будут читать его при просмотре списков багов, которые не были или не будут исправлены. Баги с неубедительными заключениями могут быть отклонены во время совещаний по отбору багов (где команда проекта решает, какие ошибки и в каком порядке следует исправлять). Руководители и другие специалисты, не входящие в команду разработки, с большей вероятностью потратят дополнительное время на баги с примечательным и аргументированным заключением. Строка резюме — ваш лучший инструмент для продажи бага этим людям.

Хорошее резюме дает читателю достаточно информации, чтобы помочь ему решить, стоит ли обращаться за пояснениями. В нем должна содержаться следующая информация:

- краткое описание, достаточно конкретное, чтобы читатель мог представить себе отказ;
- краткое указание ограничений или зависимостей бага (насколько узки или широки связанные с ним обстоятельства?);
- краткое описание влияния или последствий бага.

Всю эту информацию нельзя вписать в резюме, поскольку оно может состоять только из одной строки (около 65 символов). (Вы можете ввести и более длинный текст, но в управленческих отчетах, в которых перечислены несколько багов, обычно отображается только одна строка резюме. Остальная часть вашей записи не видна.) Выберите то, что наиболее важно для вашего отчета, а остальное оставьте для разделов с подробным описанием его содержимого.

Никогда не преувеличивайте свои баги

Ваш авторитет — основа вашего влияния. Если вы сделаете так, что баги, о которых вы сообщаете, будут казаться более серьезными, чем они есть на самом деле, вы потеряете свое влияние.

В вашей компании существует схема ранжирования багов: например, мелкие, серьезные и критические. Здесь мы не определяем степень серьезности, поскольку в разных компаниях эти определения сильно

разнятся. Какими бы ни были нормы вашей компании, работайте в их рамках. Не стоит относить к категории серьезных баг, который обычно классифицируется как мелкий, только для того, чтобы привлечь к нему дополнительное внимание. Если вы считаете, что принятая в вашей компании схема неправильно применяется к багу, о котором вы сообщаете, то используйте тот ранг, который считаете подходящим, но объясните свои действия в конце описания проблемы («Я знаю, что обычно подобная проблема классифицируется как мелкая, но считаю, что конкретно этот баг должен быть классифицирован как серьезный, поскольку…»).

Сообщайте о проблеме недвусмысленно, но не пытайтесь ее решить

Ваша работа — сообщать о проблемах, а не выявлять первопричины и не добиваться конкретных решений.

Не изучив базовый код, вы не сможете определить первопричину отказа. Слишком часто мы видели отчеты, в которых основное внимание уделялось (неверной) теории тестировщика о причине, а фактических данных было недостаточно для того, чтобы программист мог легко понять, что же на самом деле увидел тестировщик.

Некоторые программисты склонны отклонять отчеты, «решения» которых, по их мнению, являются несостоятельными, не рассматривая при этом основную проблему. Не делайте так, не давайте повода отклонять ваши отчеты.

Выбор решения о том, как исправить данный баг, — задача дизайнера продукта. Например, вы обнаружили сообщение об ошибке, которое исчезает, как только пользователь перемещает мышь, что затрудняет чтение этого сообщения. У вас может возникнуть соблазн написать баг-репорт следующего содержания: «Сообщение об ошибке должно появляться в диалоговом окне, которое всегда находится сверху, пока не будет явно закрыто». Если вы так поступите, то не удивляйтесь, если в ответ дизайнер попросит вас не лезть не в свое дело. Потому что существуют разные способы вывода сообщений об ошибках. Дизайнер выбирает тот, который ему кажется наиболее подходящим.

Другая сложность, связанная с баг-репортами, ориентированными на решение проблем, заключается в том, что многие тестировщики настолько увлекаются своим предлагаемым решением, что не предостав-

ляют чистую и точную информацию о самом отказе. Если тестировщик неправильно понимает основную причину отказа, то предлагаемое им решение в лучшем случае бесполезно. Зачастую оно оказывается хуже чем бесполезным, поскольку тестировщик непреднамеренно опускает ключевую информацию или пишет отчет, который привлекает внимание программиста к неверным деталям. Такие отчеты иногда вызывают много смеха у программистов. Возможно, вы не захотите быть тем человеком, над чьими невежественными предложениями они смеются.

Сообщить о проблеме с исчезновением сообщения можно куда более удачным образом: «Появилось сообщение об ошибке, но я не смог его прочитать, поскольку оно исчезло, как только я переместил мышь». Если у вас хорошие отношения с дизайнером, то можете добавить: «Если бы это было модальное диалоговое окно, то это могло бы решить проблему». Обратите внимание, что это предложение не звучит как приказ.

Если известно, что какая-то часть программы должна работать исключительно одним способом, то проблема может быть обусловлена отсутствием решения. Учитесь видеть и не переходить эту грань.

Выбирайте интонацию. Каждый человек, которого вы критикуете, увидит отчет

Баг-репорт, написанный в обвиняющем или покровительственном тоне, ни к чему хорошему не приведет. Никогда не стоит называть программиста непрофессиональным, ограниченным или глупым. Даже если в порыве эмоций хочется так сделать, сдержите себя, иначе впоследствии ваш авторитет будет подорван, за вашей работой будут наблюдать более пристально и к тому же вы с меньшей вероятностью добьетесь исправления многих багов, о которых сообщаете. Следите также за форматированием. Например, отчеты, написанные ЗАГЛАВНЫМИ БУКВАМИ, выглядят так, будто вы КРИЧИТЕ. Если вы не уверены в том, какое впечатление производит отчет, попросите кого-нибудь другого прочитать его и ВНИМАТЕЛЬНО ВЫСЛУШАЙТЕ его комментарии.

Если для вас или для некоторых программистов в вашей компании тон является проблемой, то попробуйте прочитать свой отчет вслух. Проговорите слова с угрожающей, саркастичной или безразличной интонацией. Другой способ проверить тон отчета — передать его черновик на проверку человеку, которому вы доверяете.

Сделайте свои отчеты читабельными даже для людей, которые устали и раздражены

Значительное количество багов исправляется (и откладывается) в последние недели проекта программистами, которые работают сверхурочно, чтобы завершить подготовку продукта. В сложных проектах программисты часто недосыпают, находятся в состоянии стресса и пьют очень много кофе. Пишите отчеты, которые смогут понять эти программисты.

Сделайте описание этапов воспроизведения простым.

- Пройдитесь по багу шаг за шагом.
- Пронумеруйте каждый шаг.
- Не пропускайте ни одного шага, необходимого для воспроизведения проблемы.
- Перечислите кратчайший набор шагов, которые приведут читателя отчета к отказу программы.
- Используйте отступы и пустые строки, чтобы облегчить чтение отчетов.
- Используйте короткие, простые предложения.
- Укажите, что произошло и что, по вашим ожиданиям, должно было произойти.
- Если последствия серьезны, но у вас есть основания предполагать, что программист не поймет, почему они серьезны, то объясните, почему вы считаете их серьезными.
- Добавьте комментарии, если они помогут программисту распознать проблему или облегчат вам повторное тестирование бага после его устранения.
- Для сложных продуктов или проблем используйте первые три строки описания для краткого изложения сути проблемы. Затем опишите подробности.
- Сохраняйте нейтральный тон отчета.
- И не шутите — это будет неправильно понято.

Совершенствуйте навыки составления отчетов

Изучите баг-репорты в системе отслеживания, чтобы найти идеи по улучшению отчетов. Например.

- Сравните закрытые баги, которые были исправлены или нет. Найдите различия в том, как о них сообщалось в отчетах. Если вы хотите, чтобы ваши баги были исправлены, то сообщайте о них так же, как о тех, которые были исправлены.
- Читайте ответы программистов (и других людей) на баг-репорты. Что приводит этих людей в замешательство? Что их злит? Что они считают неприемлемым? За что благодарны?

При необходимости используйте рыночные данные или данные службы поддержки

По возможности сравните поведение вашего продукта с поведением продукта ведущего конкурента. Это поможет вам описать ожидания пользователей, а менеджерам по маркетингу — оценить (с их точки зрения) серьезность проблемы.

Спросите продавцов и специалистов по сбыту, какие вопросы возникают при продаже, как демонстрируется продукт, что нужно показать клиентам и как продавцы и специалисты по сбыту хотят это сделать. Используйте их рассказы при составлении баг-репортов.

По возможности увязывайте баг, о котором вы сообщаете, с записями технической поддержки о связанных или похожих багах. Чтобы собрать эти данные, вам, вероятно, придется тесно сотрудничать со службой технической поддержки вашей компании. Если это возможно, то оцените затраты на поддержку, связанные с отсрочкой устранения бага, или те проблемы, с которыми столкнутся люди (клиенты или сотрудники технической поддержки), если он останется в продукте.

Просматривайте баг-репорты друг друга

В некоторых командах второй тестировщик просматривает сообщения о дефектах до того, как они попадут к программисту. Второй тестировщик:

- проверяет наличие и понятность критической информации;
- проверяет, может ли он воспроизвести ошибку;
- спрашивает, можно ли упростить, обобщить или усилить отчет.

Этот тестировщик может провести анализ:

- всех дефектов;
- всех дефектов в своей области;
- всех дефектов своего напарника.

Если второй тестировщик находит проблемы, то возвращает баг автору отчета.

- Если автор был тестировщиком, то проверяющий указывает на проблемы, чтобы дать ему возможность повысить квалификацию.
- Если автор не входил в команду тестирования, то второй тестировщик просто проверяет с ним основные факты.

Это полезный способ обучения персонала и уточнения отчетов, однако не перегружайте проверяющих тестировщиков. Данный процесс требует времени. Кроме того, определите, когда рецензент должен воспроизвести каждый баг, а когда — просто проверить достоверность и понятность каждого отчета.

Познакомьтесь с программистами, которые будут читать ваши отчеты

Вы с большей вероятностью напишете вежливый и продуманный отчет и оформите его так, чтобы его было легче понять, если знаете человека, который будет читать этот отчет. Если же программист анонимен, то вы скорее увидите в нем идиота, чем ошибающегося человека, который пытается хорошо выполнить свою работу.

Наилучшим подходом может быть демонстрация багов программистам

Обнаружив баг, некоторые тестировщики подходят к программисту, занимающемуся данной областью, и описывают или демонстрируют этот баг. По результатам обсуждения они могут продолжить устранение неполадок или просто написать отчет как есть. В некоторых компаниях такая практика поощряется, в других — не приветствуется. Нам она в целом нравится, но если вы еще не наладили рабочие отношения с программистом, мы настоятельно рекомендуем немного поработать с багом (сделать его воспроизводимым, возможно, провести дополнительное тестирование),

прежде чем обращаться к программисту. Чем меньше вы знаете этого человека, тем лучше должны быть подготовлены к встрече с ним.

Этот подход работает особенно хорошо при тестировании сложного продукта, когда программисту могут понадобиться данные, которые тестировщик не знает, как представить. Он может учиться у программиста, а программист имеет доступ к системе в момент отказа.

Если программист выглядит сосредоточенным и занятым, когда вы заходите к нему, то не прерывайте его работу. Вместо этого отправьте электронное письмо о том, что вы нашли интересную проблему и хотели бы поговорить с ним в ближайшее время, прежде чем вносить ее в систему отслеживания багов. Позвольте ему поговорить с вами, когда он будет готов. Если он готов слишком редко, то просто отправьте баг-репорт без него.

Когда программист скажет, что проблема исправлена, убедитесь, что это действительно так

В условиях дефицита времени многие программисты выбирают наиболее быстрый путь устранения симптомов, описанных в баг-репорте. При этом исправление может быть как полным, так и частичным. При повторном тестировании якобы исправленной программы вы можете обнаружить, что она не работает при несколько иных обстоятельствах, например при использовании других данных, или что исправление вызвало новую проблему. Проведите повторное тестирование, чтобы убедиться, что симптом не проявился в другом элементе программы.

Оперативно проверяйте исправление багов

Если вам сказали, что один из багов, о котором вы сообщили, был исправлен, то проверьте его как можно скорее. Уделяя внимание проверке исправлений, вы проявляете уважение к программисту и повышаете вероятность того, что он будет быстро реагировать на ваши баг-репорты в дальнейшем.

Если вы быстро обнаружите проблему с исправлением и быстро сообщите об этом программисту, то он, скорее всего, еще будет помнить, что делал с кодом, и сможет сразу же устранить проблему. Чем дольше вы будете ждать, тем меньше он будет помнить. (Более полезные пред-

ложения, подобные этому, об основных операциях команд тестирования см. в DeNardis, 2000.)

Если исправления не работают, то поговорите с программистом

Если исправление бага неоднократно не имеет должного результата или не работает на очень поздних этапах разработки, то не просто внесите эту информацию в баг-репорт, а сохраните ее в системе отслеживания. Обратитесь непосредственно к программисту. Если вы работаете в одном здании, то сходите в его кабинет. Если нет, то позвоните ему. (Обратите внимание: если программист работает в другой компании, то, чтобы позвонить ему, вам может потребоваться разрешение руководителя проекта. В этом случае сначала сообщите о баге этому человеку.) Ваш тон должен быть дружелюбным, а сообщение — конструктивным. Вы обращаетесь к программисту не для того, чтобы сказать: «Плохой! Плохой программист!» Вы оказываете ему услугу, сразу же предоставляя ему информацию и давая возможность прояснить любые неясности в вашем отчете и продемонстрировать баг, если он захочет его увидеть.

Баг-репорты должны закрываться тестировщиками

Если баг был отмечен как решенный, то тестировщик должен проверить его. Если баг был отмечен как исправленный, тестировщик должен попытаться показать, что исправление было неполным. Если баг-репорт был отклонен как невоспроизводимый или непонятный, то тестировщик должен исправить отчет. Если баг был отложен или отклонен как не являющийся багом, тестировщик должен решить, нужно ли собирать дополнительные данные, чтобы оспорить отсрочку или отклонение. Если баг был отклонен как дубликат, то тестировщик должен решить, согласен ли он с этим. Некоторые проектные команды скрывают баги, помечая их как дубликаты.

Как правило, тестировщик, представивший баг-репорт, проводит повторную проверку. Но если о баге сообщил не тестировщик, то этот баг должен оценить тестировщик, наиболее хорошо знакомый с данной частью программы. При возможности этот тестировщик должен также проконсультироваться с автором отчета.

Ни один баг не должен быть помечен как закрытый, если не был проверен и закрыт тестировщиком.

Не настаивайте на том, чтобы каждый баг был исправлен. Расставляйте приоритеты

Иногда существуют веские причины не исправлять некоторые баги. Одна из наиболее важных связана с риском. Каждое исправление бага (как и любое другое изменение кода) может привести к появлению новых багов. Исправляя мелкий баг, программист может создать более серьезный. Если он сделает это ближе к концу разработки проекта, то вам может не хватить времени на тестирование между изменениями и датой выпуска, которое призвано обнаружить более серьезную проблему. Испытывая опасения по поводу необнаруженных побочных эффектов, опытные руководители проекта с осторожностью относятся к внесению поздних изменений в код.

Еще одна веская причина заключается в том, что клиент может не захотеть платить за исправление. Особенно ярко это проявляется в случае заказного, контрактного программного обеспечения и собственной разработки. Клиент фактически платит за ваше время. Некоторые баги дешевле оставить, чем исправлять, и нередко клиенты готовы пойти на компромиссы, связанные со стоимостью. Представьте, что вы работаете над выпуском обновления, в котором будет исправлена критическая ошибка, приводящая к удалению данных некоторых клиентов. Программа, в которой обнаружен баг, уже используется. А теперь представьте, что выпуск обновления откладывается для исправления орфографической ошибки. Ответ клиента в случае с этим выпуском обновления кажется очевидным (не исправлять орфографическую ошибку). Что касается менее очевидных решений, то команда проекта отвечает за представление и оценку компромиссов, чтобы достичь правильного баланса между временем, стоимостью, доступностью и надежностью.

Если вы не можете доказать, что этот баг важен, или найти стейкхолдера, который был бы достаточно заинтересован, чтобы активно поддержать вашу апелляцию по поводу конкретно этой отсрочки, то мы рекомендуем оставить этот баг в покое и заняться чем-то другим.

Не позволяйте отложенным багам исчезнуть

«Отложенный» означает, что баг, о котором вы сообщили, существует, но его не исправят в текущем выпуске. Поэтому баги, которые были закрыты как отложенные в этом выпуске, становятся открытыми в начале следующего. Многие команды настраивают свои системы отсле-

живания багов таким образом, чтобы автоматически открывать старые отложенные баги или переносить их (как открытые баги) в новый файл для следующей версии продукта. Часто они также повторно открывают баги, которые были отклонены как «работает, как задумано», если соответствующие проектные решения рассматриваются заново в новом выпуске.

В продуктах, прошедших через множество выпусков, накапливается множество раздражающих баг-репортов, которые никогда не приведут к изменениям кода или улучшению документации. Некоторые команды проводят анализ вместе с руководителями проекта и навсегда закрывают некоторые баги как INWTSTA (I never want to see this again, «Я больше не хочу этого видеть»).

Проводить такой анализ лучше всего в начале проекта, когда график наименее напряженный и руководитель проекта спокоен и рассудителен.

Никогда не отказывайтесь от исправления ошибок только потому, что это усложнит тестирование

Знайте, что ваш процесс в корне неверный, если менеджер по тестированию просит программистов *не* исправлять баг (ошибку кода или конструктивную недоработку) на том основании, что это изменение затронет слишком много чек-листов, сценариев или других артефактов тестирования и, следовательно, управление этим изменением займет слишком много времени.

Немедленно обжалуйте отсрочки по багам

Если найденный вами баг отложили или отклонили (программа работает, как задумано), то решите, стоит ли обжаловать это решение. В одних компаниях подача апелляций допускается на совещаниях проектной группы (очистка багов, совещания по выбору багов и т. п.) как обычная часть процесса. В других компаниях апелляции подаются на личных встречах тестировщика или менеджера по тестированию с руководителем.

Если вы решили обжаловать отсрочку или отказ, то сделайте это как можно скорее. Если же вы хотите критиковать решение спустя несколько месяцев после его принятия, но для этого нет веских объективных оснований, то ваша инициатива не встретит понимания.

Решив бороться, стремитесь победить!

Если вы подаете апелляцию, то не полагайтесь на слова и информацию, содержащиеся в вашем первоначальном баг-репорте. Он был неубедительным. Если вы не сформируете более эффективную аргументацию, то не только потратите время, но и подорвете свой авторитет.

Чтобы подготовить апелляцию, вы можете выполнить следующие действия.

- Поговорите с другими стейкхолдерами, такими как представители службы технической поддержки, отдела документации, отдела продаж и т. д. Выясните, бюджет какого отдела пострадает больше всего, если баг останется в продукте, во сколько это обойдется отделу и насколько это беспокоит его руководство.
- Проведите дополнительное тестирование, выявляя более серьезные последствия бага или пытаясь показать, что он возникает при более широком диапазоне обстоятельств, чем описано в отчете.
- Разработайте несколько сценариев, историй, иллюстрирующих, как грамотный пользователь может столкнуться с багом при корректном использовании продукта.
- Проверьте, не обсуждалась ли в прессе проблема, подобная той, которую вы описываете. Возможно, один из ваших конкурентов выпустил продукт с аналогичным багом. Если об этом говорили во *всех* новостях, это веское доказательство того, что к багу следует отнестись серьезно.

Принцип, который мы призываем вас принять, заключается в том, что каждое ваше обращение должно быть убедительным. Даже если вы не выиграете каждую апелляцию (а неудачи, конечно, будут), вы должны создать себе репутацию человека, каждая попытка которого *заслуживает* победы.

Глава 5

Автоматизированное тестирование

Роботы, которые готовят завтрак. Летающие автомобили. Это научная фантастика. Но программное обеспечение может делать *все*. Так почему бы не создать ПО, которое будет тестировать другое ПО? Аргументация такова: если один компьютер может выполнить работу трех миллионов математиков, делающих вычисления палочками на песке, то, несомненно, один компьютер стоит целой армии тестировщиков-людей. Действительно, автоматизация тестирования — интересная и многообещающая идея. Но будьте осторожны. Полезной может оказаться автоматизация лишь некоторых видов тестирования. Она может сэкономить время, ускорить разработку, расширить сферу применения и сделать тестирование более эффективным. Или же может отвлекать вас и вынуждать тратить ресурсы впустую.

Ваши инвестиции в автоматизацию тестирования ценны в той мере, в которой помогают вам достичь поставленной цели. Роль тестирования заключается в получении информации. Какую информацию можно получить с помощью вашей автоматизации?

Одни команды благодаря автоматизации добились впечатляющих успехов. Другие — остались недовольными и разочарованными, при этом часть из них обманули себя и свое руководство, решив, что работа по автоматизации принесла пользу.

Используйте автоматизацию, если она способствует достижению цели тестирования. Оценивайте успешность автоматизации с точки зрения того, насколько она помогла вам достичь вашей цели.

Рецензенты ранних вариантов этой главы отдельно просили нас выделить несколько противоречивых моментов.

- **Прежде чем решать, какие тесты автоматизировать, сначала спроектируйте их.** Это позволит вам не попасть в ловушку автоматизации тестов, которые легко автоматизировать, но которые не способны находить дефекты.
- **Проектирование автоматизированных тестов отличается от ручного тестирования.** Бóльшая часть возможностей автоматизированного тестирования связана с использованием компьютера для выполнения тех задач, которые не под силу человеку. Ищите такие возможности, как, например, повторение одних и тех же тестов для тысяч различных файлов данных. Это позволит вам не попасть в ловушку автоматизации тестов только на основе существующих (ручных) планов тестирования и не упустить большие возможности для автоматизации тестирования. При разработке ручных тестов вы вряд ли будете рассматривать тесты, в которых применяются повторяющиеся операции над тысячами файлов, — это будет слишком трудоемко.

Эти противоречивые моменты вытекают из двух важных уроков:

- автоматизация без хорошего проектирования тестов может привести к большому количеству действий, но малой пользе;
- если проектировать тесты и при этом не иметь хорошего понимания возможностей автоматизации, то можно упустить самые ценные из них.

Мы считаем, что вы достигнете успеха, если в вашу команду будут входить хорошие дизайнеры тестов *и* хорошие автоматизаторы, которые станут участвовать в выборе и разработке ваших автоматизированных тестов. Нам легко говорить об этом. Сложнее воплотить это в жизнь.

Ускорьте процесс разработки, вместо того чтобы пытаться сэкономить несколько долларов на тестировании

Усилия по автоматизации, направленные на снижение затрат на тестирование, редко получают должное внимание и содействие, необходимые для достижения успеха. Если вы хотите получить поддержку, то сосредоточьтесь на снижении риска неудачной разработки.

Тестовые случаи эффективны в той мере, в какой помогают вам и вашей команде получить полезную информацию о тестируемой системе. Автоматизированное тестирование повышает эффективность, помогая быстро собирать и распространять информацию, позволяя программистам быстро давать обратную связь. Наиболее успешные компании автоматизируют тестирование, чтобы повысить гибкость своей разработки. В частности, они стремятся:

- быстро обнаруживать дестабилизирующие изменения в новых сборках;
- как можно быстрее выявлять регрессии багов;
- сообщать о проблемах быстро, поскольку это облегчает их устранение.

Быстрые исправления позволяют сохранить стабильность кода. Она, в свою очередь, позволяет экономить время (несколько человек не тратят его на один и тот же баг) и облегчает рефакторинг и другие работы по улучшению структуры кода и устранению спагетти. Если кодовая база в целом стабильна и имеется мощный набор автоматизированных тестов, то программисты могут приступать к внесению более серьезных изменений с меньшим риском. Кроме того, команда проекта может оперативно корректировать объем продукта и сроки его выпуска, реагируя на конъюнктуру рынка.

Приведем два примера техник поддержания темпов разработки.

- **Автоматизированные смоук-тесты.** Выражение «смоук-тестирование» пришло из области тестирования аппаратуры. Вы ставите новую плату и включаете питание. Если вы видите, что из платы идет дым, то отключите питание. Дальнейшее тестирование не требуется. Смоук-тесты (они же *тесты верификации сборки*) проверяют широкий спектр функций продукта за ограниченное время — обычно за обед или за ночь. Если ключевые функции не работают или ключевые баги еще не исправлены, то ваша команда не будет тратить дополнительное время на установку и тестирование сборки. Исправление проблемы становится главным приоритетом программистов.
- **Автоматизированные модульные тесты.** Эти тесты также позволяют оптимизировать процесс разработки, предотвратить возврат и сохранить динамику развития. Это более крупные наборы тестов, которые направлены на проверку низкоуровневых функций и классов вашего продукта.

Наибольшая ценность автоматизированных модульных и смоук-тестов заключается в том, что они могут выполняться в любое время и любым

человеком. Они запускаются автоматически в процессе сборки. Их наличие помогает отдельным программистам создавать мини-сборки, содержащие только одно или несколько изменений. Если одна из таких сборок окажется неисправной, то программист будет знать, что нужно проверить. Если с мини-сборками все в порядке, то более широкая сборка, в которой собраны все изменения, скорее всего, будет работать. Это неоценимое преимущество, которое наверняка оценит ваш руководитель проекта.

Создание таких автоматизированных тестов требует времени, усилий, навыков и денег. Модульные тесты, скорее всего, будут создаваться программистами, хотя вы можете поощрить и ускорить эту работу, написав код вместе с ними в паре. Благодаря таким преимуществам вам будет гораздо легче обеспечить необходимое сотрудничество, чем если бы вы сосредоточились на экономии времени при ручном тестировании (Beck, 1999, глава 20).

Расширяйте свои возможности, вместо того чтобы пытаться повторять одни и те же тесты снова и снова

Используйте автоматизацию, чтобы расширить свои возможности и восприятие, что позволит вам видеть больше и делать больше.

Одни тесты просто невозможно запустить без автоматизации. Другие можно выполнять в гораздо бóльших масштабах. Приведем несколько примеров.

- **Нагрузочные тесты.** Что произойдет, если 200 человек одновременно попытаются использовать ваше программное обеспечение? А если 2000? Для моделирования этих сценариев вам потребуется автоматизация.
- **Контрольные показатели производительности.** Улучшается или ухудшается производительность системы? Автоматизированные тесты можно оснастить инструментами, позволяющими проводить измерения времени при каждом запуске. Собирая эти измерения и анализируя их в виде временно́го ряда, можно обнаружить снижение производительности. Аналогичный подход можно использовать для оценки использования ресурсов, таких как память или хранилище.
- **Тестирование конфигураций.** Программное обеспечение часто должно работать на различных платформах, в разных конфигурациях, подключаться к различным периферийным устройствам. Как покрыть

их все? Автоматизация помогает увеличить покрытие тестов. Для этого необходимо обеспечить их переносимость на разные платформы.

- **Тестирование на выносливость.** Что произойдет, если ваш продукт будет использоваться в течение нескольких недель или месяцев? Утечки памяти, повреждения стека, недействительные указатели и другие ошибки могут быть незаметны в момент возникновения, но в конечном итоге приведут к проблемам. Одна из стратегий заключается в выполнении серии тестовых случаев в течение длительного периода времени — дней или недель — без перезагрузки системы. Это требует автоматизации.
- **Состояния гонки.** Некоторые проблемы возникают только в определенных временны́х ситуациях. Совпадение по времени двух потоков или процессов, конкурирующих за один и тот же ресурс, приводит к ошибке, называемой *состоянием гонки*. Часто такие ошибки трудно найти и трудно воспроизвести. Автоматизация может оказаться весьма полезной, поскольку можно повторить тесты с множеством слегка различающихся временны́х характеристик.
- **Ошибки комбинирования.** Некоторые ошибки связаны со взаимодействием нескольких функций. С помощью автоматизации можно протестировать огромное количество сложных тестов, в каждом из которых по-разному используется несколько функций.

Эти подходы направлены на использование автоматизации для создания новых тестов или для повторного применения продукта в целях выявления новых багов. Ни один из этих тестов не является простым в реализации. Возможно, вам придется работать над ними, автоматизируя различные этапы тестирования и разрабатывая вспомогательные инструменты. Тем не менее мы считаем, что зачастую это более эффективная цель действий по автоматизации, чем простое повторение одних и тех же тестов.

Выберите стратегию автоматизации в зависимости от своего контекста

Стратегии автоматизации тестирования зависят от требований к тестированию, архитектуры программного продукта и квалификации персонала, занимающегося тестированием.

- **Тестирование требований.** Программные продукты могут иметь множество функций, но ключевыми зачастую являются лишь некоторые

из них. Они должны быть надежными. Они могут требовать всестороннего тестирования, что оправдывает усилия по автоматизации. В качестве альтернативы можно сосредоточиться на том, как автоматизированное тестирование может помочь в управлении основными рисками продукта. Для проверки других аспектов вашего продукта, возможно, будет достаточно ручного тестирования.

- **Архитектура программного продукта.** Проанализируйте архитектуру продукта, чтобы определить возможности автоматизации тестирования. Каковы основные компоненты программного обеспечения? Как они взаимодействуют между собой? Какие технологии используются? Каковы доступные точки соприкосновения? Архитектура описывает программные компоненты и их связи с различными частями системы. Языки, среды и компоненты влияют на то, какие средства тестирования подходят для той или иной задачи. Используемые интерфейсы определяют возможности автоматизации (Hoffman, 1999b).
- **Квалификация тестировщиков.** Одни подходы к автоматизации хороши тем, что позволяют задействовать тестировщиков, не занимающихся программированием. Другие в полной мере используют навыки тестировщиков-программистов. Однажды одного из нас (Петтикорда) попросили порекомендовать единый подход к автоматизации тестирования для компании с множеством продуктов, разрабатываемых в разных местах. Брет не смог этого сделать. В одних местах в основном работали тестировщики, умеющие программировать. В других — тестировщики, которые этого не умели. Единый подход не мог подойти для всех. Несмотря на схожесть продуктов, специализация и возможности сотрудников были слишком разными.

Не требуйте стопроцентной автоматизации

Некоторые руководители программных продуктов пребывают в заблуждении, что автоматизированное тестирование *всегда* равно лучшему тестированию, и требуют, чтобы автоматизировано было все тестирование. Такие требования нанесли ущерб многим проектам.

Если ваш руководитель доверчив, то, возможно, вам или другому старшему тестировщику следует постараться быть рядом, когда ему демонстрируют новейший инструмент тестирования или читают лекцию о новейшей методологии тестирования, представляя его/ее как универсальное решение.

Скептически относитесь к заявлениям консультантов или продавцов инструментов о том, что ведущие компании обычно достигают

100%-ной автоматизации. Потребуйте доказательств. Многие ведущие компании добиваются успеха, комбинируя техники тестирования, в том числе неповторяющееся минимально документированное исследовательское тестирование. И так и должно быть. Автоматизация тестов обычно означает более частое выполнение меньшего количества тестов. Многие тесты стоит выполнять только один раз.

Есть еще один миф, который необходимо развеять: якобы тестирование происходит путем нажатия одной кнопки, при этом о найденных багах сообщается автоматически. Поставщики инструментов, предлагающие интеграцию между своими средствами тестирования и отслеживания, поощряют эту безответственную фантазию. Мы никогда не видели, чтобы такая интеграция улучшала тестирование, отчетность или отслеживание багов. Если у вас нет людей, которые анализируют отказы до того, как о них будет сообщено, то вы потратите время программистов и вызовете их недовольство. Они будут возмущены тем, что им приходится разбираться с множеством ложных срабатываний и дубликатов.

Тем не менее в определенных ситуациях цель создать максимально большой и фундаментальный набор автоматизированных тестов может быть оправданной.

- По договорным или нормативным причинам необходимо доказать, что окончательная версия продукта прошла серьезную проверку.
- Вы будете поставлять вместе с продуктом тесты, которые клиенты смогут выполнять самостоятельно. (Такая практика широко распространена в телекоммуникационной отрасли.)
- К вашему продукту предъявляются жесткие требования по обратной совместимости, которые сохранятся на протяжении многих версий.
- Единственный способ протестировать *некоторые* продукты — написать программы для их использования.

Однако даже в этих случаях, на наш взгляд, у тестировщиков всегда должна быть возможность проведения неофициальных ручных или одноразовых, недокументированных тестов. Не накладывайте искусственных ограничений на тестирование.

Инструмент тестирования — это не стратегия

Инструмент тестирования не научит ваших тестировщиков тестированию. Если в процессе работы возникает путаница, то инструменты

будут ее усиливать. Исправьте свои процессы тестирования, прежде чем автоматизировать их.

Некоторые инструменты тестирования поставляются в комплекте с элементарными рекомендациями по стратегии тестирования. Однако эти рекомендации редко бывают достаточно информативными, не учитывают особенности конкретной ситуации и, как правило, преувеличивают важность своего типа автоматизации.

Нам приходилось общаться с руководителями, которые считали, что вместо того, чтобы нанимать тестировщиков, они могут купить инструмент тестирования. (Мы встречали поставщиков инструментов, которые продвигали эту идею.) Мы никогда не видели, чтобы это сработало. Если вы купите такой инструмент, то, возможно, избавитесь от тестировщиков, но в итоге получите кучу занятых пользователей инструмента, много активности и мало (или ноль) обнаруженных багов. Где в этом ценность?

Не автоматизируйте беспорядок

Если у вас плохие тесты, автоматизация поможет вам быстрее провести плохое тестирование. Если тестирование неорганизованное, то проект автоматизации погрязнет в путанице. Если ваши тестировщики не знают или не хотят говорить, что они тестируют, то будут создавать автоматизацию, правила использования которой никто не сможет понять.

Вот сценарий, с которым сталкивались мы и другие консультанты: компания привлекает консультанта для автоматизации тестирования своего программного обеспечения. Первоначально проблема заключается в том, чтобы заставить инструмент тестирования графического интерфейса пользователя работать с тестируемым ПО. Это требует затрат времени и денег, но в итоге проблема решается. По мере того как проект автоматизации продвигается все дальше, консультант понимает, что основная проблема заключается в отсутствии процесса тестирования. Поэтому инструмент откладывается на потом, и усилия сосредотачиваются на устранении основной проблемы. Только тогда начинают проявляться положительные результаты.

Если ваше тестирование неорганизованное, то автоматизация будет отвлекать вас от того, что вам действительно нужно делать.

Не приравнивайте ручное тестирование к автоматизированному

При выполнении теста вручную задействуется весь спектр человеческих возможностей. Вы можете импровизировать, создавая новые тесты, или заметить то, что не предусмотрели или не могли предугадать. Автоматизация тестирования — слабый отголосок этого сложного интеллектуального процесса. Вот почему бессмысленно говорить об автоматизированных тестах так, будто они являются автоматизированными тестами, проводимыми тестировщиком-человеком.

Автоматизация не заставляет компьютер проводить то тестирование, которое делаете вы. Он выполняет то тестирование, которое вы явно определяете, не имея возможности воспользоваться вашими неявными знаниями и компетентностью. Автоматизированный тест делает одно и то же при каждом запуске, с одной и той же скоростью, в одном и том же порядке, с одними и теми же движениями мыши и щелчками клавиш. Тестировщик же, выполняющий тестирование вручную, не может не изменять тест при каждом запуске. Эти вариации могут выявить невидимые баги.

Автоматизированная проверка результатов также имеет свои ограничения. Возьмите любую процедуру тестирования. Говорится ли в ней о том, что нужно прислушаться к странным звукам, доносящимся из динамика? Нужно ли искать необычное мерцание на экране? Нужно ли сообщать о постепенном снижении производительности? Скорее всего, нет. Тем не менее, даже не имея инструкции, хороший тестировщик заметит такие проблемы и сообщит о них. Подготовленный разум — фантастический инструмент тестирования, превосходящий все мыслимые средства автоматизации. Ваш разум может заметить сотни проблем, о которых вы даже не подозреваете, пока не скажете: «О, что это? Что-то тут не так».

Автоматизация имеет много преимуществ, но увеличивает расстояние между человеком-тестировщиком и тестами. Это означает, что проблемы могут возникнуть и при этом остаться незамеченными. В отличие от автоматизированных тестов тестировщик-человек может сразу же исследовать аномалии, сохраняя контекст их возникновения. Кроме того, он может распознать и пропустить как допустимый широкий спектр ложных срабатываний, которые могут сбить с толку автоматику.

Поэтому не сравнивайте ручные тесты с автоматизированными. Вместо этого рассматривайте автоматизированные тесты как средство,

позволяющее расширить ваши возможности, сделать то, что вы не смогли бы сделать при ручном тестировании.

Не судите о ценности теста по частоте его проведения

Ценность теста определяется информацией, которую он предоставляет. Оценить это сложно. Квалифицированное тестирование — это зачастую вопрос правильного применения таких суждений.

Некоторым тестировщикам советуют попытаться оценить, обеспечит ли автоматизация возврат инвестиций, сравнив затраты на автоматизированные тесты с затратами на выполнение тех же тестов вручную (см., например, Linz, Daigl, 1998a и 1998b; Dustin et al., 1999, 52; и Fewster, Graham, 1999, 519).

Мы считаем, что при таком подходе измеряются не те факторы и не теми способами (Hoffman, 1999a).

Вот те уравнения, которые, по нашему мнению, в корне неверны:

$$\text{стоимость ручного тестирования} = \\ = \text{стоимость ручной подготовки} + \\ + (N \times \text{стоимость ручного выполнения});$$

$$\text{стоимость автоматизированного тестирования} = \\ = \text{стоимость автоматизированной подготовки} + \\ + (N \times \text{стоимость автоматизированного выполнения}),$$

где N — количество запусков теста.

В качестве простого приближения затрат эти формулы вполне справедливы. Они отображают общее наблюдение о том, что автоматизированное тестирование обычно требует более высоких первоначальных затрат, но при этом снижает стоимость выполнения.

У нас есть два принципиальных возражения против использования этих формул для обоснования усилий по автоматизации.

1. Сами тесты несравнимы. *Не приравнивайте ручное тестирование к автоматизированному* (см. урок 108). Они просто не дают одинаковой информации.
2. Не имеет смысла сравнивать стоимость выполнения автоматизированного теста, скажем, 50 раз со стоимостью выполнения того же теста вручную 50 раз. *Кто будет выполнять одни и те же тесты*

50 раз вручную? Это не будет стоить затраченных усилий. Ценность теста — информация, которую он предоставляет, — не оправдывает такие действия!

Да, автоматизация позволяет проводить тесты чаще, но вы экономите деньги каждый раз, когда проводите их, только если действительно выполняли бы их вручную.

В любом случае, принимая решение об автоматизации тестов, необходимо провести анализ затрат и выгод. Если он проведен неправильно, то и ресурсы будут распределены ненадлежащим образом.

Автоматизированные регрессионные тесты находят меньшую часть багов

Неофициальные опросы показывают, что процент багов, найденных автоматизированными тестами, на удивление низок. В проектах с серьезными, хорошо продуманными усилиями по автоматизации регрессионные тесты находят около 15 % от общего количества багов, о которых сообщается (Marick, online).

Автоматизаторы регрессионных тестов, как правило, находят больше багов в процессе разработки тестов, чем при их последующем выполнении. Однако если вы найдете возможность повторно использовать свои регрессионные тесты в различных средах (например, на другой аппаратной платформе или с другими программными драйверами), то ваши тесты с большей вероятностью найдут проблемы. По сути, это уже не регрессионные тесты, поскольку они тестируют конфигурации, которые ранее не проверялись. Тестировщики отмечают, что такие автоматизированные тесты позволяют сэкономить от 30 до 80 % средств и времени.

Новые тесты старых функций с такой же вероятностью могут найти регрессии багов, как и старые тесты. К тому же у них есть дополнительное преимущество — возможность найти баги, которые были в программе с самого начала[1].

---

[1] Есть два исключения. 1. Если проекту вредит плохое управление конфигурацией, то в него может быть повторно внедрен старый код, который принесет с собой старые баги. Однако правильное управление конфигурацией — обычно более дешевое и эффективное решение, чем регрессионное тестирование. 2. Плохо разработанный код может привести к тому, что локальное исправление одной проблемы приведет к возникновению другой. Когда вторая проблема впоследствии обнаруживается и исправляется, первая появляется вновь.

Подумайте, какие баги вы не обнаружите, пока автоматизируете тесты

При расчете стоимости автоматизации мы предлагаем обратить внимание на альтернативные издержки. Что еще вы могли бы сделать за то время, которое тратите на автоматизацию? Какие тесты вы не выполните? Какие ошибки не найдете (Marick, 1998)?

Непосредственным следствием многих проектов автоматизации является задержка тестирования и поиска багов. Эта проблема также является аргументом в пользу использования методов предварительной автоматизации, которые позволяют подготовить автоматизированные тесты до того, как ПО будет готово к тестированию. Например, стратегия, основанная на данных, может позволить заранее определить тестовые входные данные. В этом случае автоматизация тестирования может помочь ускорить тестирование и поиск багов.

Проблема плохой автоматизации заключается в том, что эту «плохость» никто не замечает

Продукты, которые уже существуют какое-то время, часто имеют автоматизированные наборы тестов. Созданные в прошлом, они до сих пор используются для тестирования продукта.

Тестировщики часто считают, что им повезло иметь такие тестовые наборы. Мы хотели бы сделать несколько предупреждений.

- **Тесты могут делать не то, что вы предполагали.** Как узнать, что они выполняют то, что должны? Автоматизаторы иногда пишут процедуры тестирования, отличающиеся от тех, которые были им поручены. Когда-то давно у них могли быть веские причины для этого, а может быть, они просто запутались. Как вы узнаете об этом? Насколько легко просматривать тесты и читать код?
- **Тесты могут перестать быть интересными.** Рецензент Дуглас Хоффман сообщает: «Недавно я обнаружил автоматизированный тест памяти, используемый одним из производителей компьютеров, который назывался “тестом одинокого бита”. Это был тест, позволявший находить определенные ошибки в разводке памяти на магнитных сердечниках, использовавшейся до 1970-х годов. Самые большие блоки такой памяти имели объем 16 Кбайт, а тесты выполнялись в течение нескольких минут. В современных устройствах

с гигантскими объемами памяти они могут выполняться часами. Ошибка, которую ищет этот тест, невозможна в современных системах памяти. Еще один автоматизированный тест, анализ которого я проводил недавно, был разработан для проверки конечного автомата, реализованного на основе встраиваемой процессорной системы. Я увидел, что документация по этому тесту была написана в 1986 году, семь поколений продуктов назад. И нет, никто не подумал обновлять тесты, когда конечные автоматы менялись каждое поколение».

- **Покрытие может быть плохим.** Люди часто обращают внимание на количество тестов в наборе. Размер набора легко увеличить, добавив множество мелких вариаций одного и того же теста. Проверьте фактическое покрытие кода этим набором. Вас могут удивить размеры областей, не охваченных тестами.
- **Ложные срабатывания могут быть обычным явлением.** Со временем небрежное отношение к автоматизации и сопровождению может привести к тому, что многие тесты будут завершаться неудачно из-за багов автоматизации. Конечно, вы не можете знать наверняка, какие отказы будут являться ошибками продукта, а какие — ошибками автоматизации, поэтому вам придется исследовать их все. Многие из них могут оказаться сломанными уже давно, потому что ни у кого не было времени их починить. Удалите сломанные тесты из своих наборов. Что в них хорошего? Они бесполезны. Примите это и удалите их или почините. Но не оставляйте их в активных наборах тестирования.
- **Результаты тестов могут быть ложными.** Одни тесты не проверяют никаких результатов. Они проходят, если продукт не выходит из строя. В других тестах есть баги, которые могут привести к пропуску части тестов или не позволят сообщить об обнаруженных отказах. Еще одни тесты имеют недопустимые «золотые» выходные файлы: если тест завершается неудачно, то вместо сообщения о баге происходит обновление «золотого» выходного файла. Мы это не выдумываем. По нашему опыту, подобные проблемы можно ожидать при использовании относительно сложных наборов тестов. Чтобы избежать их, автоматизаторы должны упрощать задачу или использовать методы безопасного программирования. Такие проблемы достаточно серьезны, но мы также видели и слышали о многих случаях, когда тесты были просто жестко запрограммированы на прохождение. Они просто говорят: `result=pass`. Автоматизатор не мог даже *подумать*, что они могут сработать. Мы предполагали, почему

так происходит, но не нашли убедительных объяснений[1]. Все, что мы можем сказать, — это то, что такое случалось в самых разных местах. Мы постоянно слышим новые истории от людей, которые видели, как это происходит. Что вы думаете о тестах, которые проходят успешно, даже если продукт, который они должны тестировать, не существует в тестируемой системе?

Хорошие наборы тестов живут и развиваются. В них добавляются новые тесты, а старые исправляются или удаляются. Если этого не происходит, то ваш набор тестов начинает костенеть. Вскоре его создатели займутся чем-то другим, а набор начнет приобретать статус мифического величественного старого дуба, к которому обращаются за советом герои мультфильмов. С возрастом дерево каким-то образом приобретает статус и репутацию. Идея о том, что старый мудрый дуб должен быть прав, потому что существует давно, может быть подходящей для героев мультфильмов, но мы считаем, что тестировщики должны быть более скептичными. Мы называем синдром веры в мудрость и опыт предков-тестировщиков, создавших грандиозные (теперь уже старые) наборы тестов, *синдромом старого дуба*.

Учитывайте возможный сбой воспроизведения записи

К наиболее популярным инструментам тестирования относятся регистраторы. На первый взгляд они предлагают простой способ создания автоматизированных тестов без какого-либо программирования. Достаточно включить регистратор во время выполнения тестов. Он фиксирует каждое событие пользователя (каждое нажатие клавиши, движение мыши и щелчок ее кнопками) и сохраняет их в сценарии для последующего воспроизведения. Кроме того, необходимо указать контрольные точки. Это этапы тестирования, на которых при нажатии горячей клавиши информация на экране сохраняется вместе со сценарием. Во время повтора сценарий сравнивает текущий экран (или некоторые связанные с ним программные атрибуты) с сохраненной информацией. Если они различаются, то тест завершается неудачей.

Основная проблема заключается в том, что сценарии слишком жестко привязаны к второстепенным деталям пользовательского интерфейса

[1] Мэри Суини сообщает о случаях, когда это происходило из-за того, что автоматизаторы начинали работу с шаблоном, содержащим строку results=pass, а затем по ошибке забывали ее удалить.

и конфигурации системы. На практике тесты часто терпят неудачу как из-за запланированных изменений в дизайне продукта, так и из-за непреднамеренных регрессий багов. Тестировщики обнаруживают, что на анализ неудачных тестов и их перезапись тратится больше времени, чем на выполнение тестов вручную.

Предположим, у вас есть 100 тестовых сценариев, содержащих создание учетной записи. Предположим, продукт изменился. Теперь в рабочем процессе создания учетной записи возникнет дополнительное диалоговое окно. Все эти тесты потерпят неудачу. Придется записывать их заново. Или предположим, что метка на экране переименована с «Регистрационное имя» на «Имя пользователя». Теперь все тесты, содержащие контрольные точки для этого экрана, завершатся неудачно.

Когда тесты не работают, их бывает очень сложно проанализировать. В чем причина: в баге настройки, автоматизации или продукта? Трудно сказать, глядя на сценарий, который был сгенерирован в процессе работы. Если вы не можете прочитать сценарий, то как определить, как он должен работать?

Более структурированные стратегии автоматизации тестирования используют те же инструменты, но их разработка занимает больше времени и требует навыков программирования. Полученные тесты гораздо проще обновлять при изменении пользовательского интерфейса.

Наш опыт, как и опыт большинства автоматизаторов, показывает, что автоматизация тестирования на основе воспроизведения записей сродни ледяному склону с кирпичной стеной внизу. Слишком много команд тратят большое количество времени на запись автоматизированных тестовых сценариев, обнаруживают, что они ломаются уже после нескольких сборок, и записывают их заново.

Инвестируйте в навыки и планирование, необходимые для создания автоматизированных тестов, которые будут работать долго, или перейдите на ручное тестирование. Любой из этих вариантов обычно оказывается более эффективным и действенным, чем захват и воспроизведение.

(Мы понимаем, что средства захвата и повтора полезны для изучения инструментария и помогают создавать тестовые сценарии вручную. Мы возражаем против того, чтобы рассматривать их как самостоятельное решение.)

Инструменты тестирования полны багов

Тестировщики возмущаются, когда понимают, что качественные инструменты, купленные по серьезной цене, сами полны багов. Действительно, инструменты тестирования часто содержат больше багов, чем аналогичные (но более дешевые) средства разработки. Планируйте тестирование своих инструментов и тратьте время на поиск способов, позволяющих устранять баги.

Некоторые инструменты предназначены для поддержки тестирования конкретных компонентных технологий. Отсутствие тестируемости в этих технологиях приводит к одному из источников багов в инструментах тестирования.

- Программисты инструментальных средств должны сначала дождаться появления этой технологии.
- Затем они должны провести реинжиниринг, чтобы понять, как добавить поддержку тестирования.
- В это время программисты вашего продукта начинают использовать новейшую компонентную технологию. Вы расстроены тем, что поставщик инструментальных средств еще не выпустил необходимое обновление. Поставщик торопится с тестированием, чтобы успеть выпустить его в нужный момент.
- Сюрприз! Спешная работа приводит к появлению множества багов.

Этот цикл будет продолжаться до тех пор, пока тестируемость не будет внедряться в компонентную технологию, а не разрабатываться с самого начала. Мы надеемся на улучшение ситуации и слышим обещания, но трудно сказать, когда это начнет происходить. Это общеотраслевая проблема.

Инструменты тестирования иногда функционируют некорректно из-за багов в других компонентах. Один из нас, Петтикорд, имел дело с инструментом выполнения, который работал ненадежно. Иногда не генерировались события мыши. В конце концов мы отследили проблему до бага в драйвере операционной системы. Ненадежность можно было воспроизвести и без инструмента. Почему тестировщики, работающие вручную, не нашли ее? Потому что они не замечали, что случайное движение мыши или щелчок ее кнопками не регистрируется. Даже если они замечали это, то, скорее всего, винили себя или аппаратное обеспечение мыши. Мы сообщили об этом поставщику операционной системы, но, похоже, его это не обеспокоило. Видимо, серьезное влияние

на тестируемость не имело большого значения. В конце концов мы решили эту проблему, используя другие драйверы мыши[1].

Другие инструменты могут настолько повлиять на тестируемый продукт, что сделают его непригодным для использования. Мониторы покрытия и памяти должны оснащаться программным обеспечением и занимать значительный объем памяти. Такие инструменты могут замедлить работу продукта до такой степени, что тестирование станет невозможным.

Поскольку инструменты тестирования имеют такую плохую репутацию, ваши программисты, скорее всего, потребуют, чтобы вы воспроизводили каждый баг, найденный с помощью автоматизации, за пределами инструмента тестирования, прежде чем воспримут его всерьез. Это еще больше усложняет анализ отказов автоматизированных тестов.

Пользовательские интерфейсы меняются

При автоматизации тестирования графического пользовательского интерфейса (GUI) значительную сложность представляет постоянное слежение за изменениями его дизайна. Если вы автоматизируете GUI, предусмотрите это.

Сказать программистам, что работу над пользовательским интерфейсом нужно заморозить, не получится. Ранние версии интерфейсов часто нуждаются в доработке. Не ставьте себя в положение человека, рекомендующего отказаться от улучшений. GUI все равно будет меняться.

Абстрагируйте интерфейс в проекте автоматизации тестирования. При изменении интерфейса вы будете обновлять слой абстракции, а не каждый тест, обращающийся к измененному интерфейсу.

Вот несколько приемов, позволяющих абстрагироваться от GUI продукта.

- **Карты окон.** Средства тестирования GUI поддерживают различные методы идентификации элементов управления окнами, такие как

[1] Боб Джонсон сообщает об аналогичной проблеме с использованием другой технологии: «Драйвер мыши действительно не играл никакой роли в требуемом тестировании нашего приложения. Однако в целях автоматизации мы должны были отследить эту проблему, и в нашем случае это заняло несколько дней времени как тестировщиков, так и разработчиков. Это несколько тысяч долларов скрытых затрат на автоматизацию».

внутренние имена, различные свойства, смежные метки и порядковая позиция. Вместо того чтобы встраивать метод идентификации в каждую ссылку на элемент управления, используйте карты окон, чтобы связать имя с методом идентификации элемента управления. Если при изменении пользовательского интерфейса необходимо изменить описание метода, то достаточно обновить карту окна. Некоторые инструменты тестирования поддерживают создание и использование карт окон, которые также называются *картами GUI* или *декларациями окон*. Если ваш инструмент не имеет встроенной поддержки, то обычно ее можно создать без особых проблем. Карты окон обеспечивают поддержку небольших изменений GUI, таких как переименование меток или перемещение элементов управления по экрану. Кроме того, карты могут применяться для поддержки повторного использования тестов с интерфейсами, переведенными на другие языки.

- **Автоматизация тестирования на основе данных.** (См. урок 127 «Автоматизация тестирования на основе данных упрощает запуск множества вариантов теста».) Этот метод обеспечивает некоторую абстракцию: вы должны иметь возможность изменять процедуру тестирования и при этом использовать тестовые данные, которые создали для этого.
- **Библиотеки задач.** (См. урок 126 «Не создавайте библиотеки тестов только для того, чтобы избежать повторения кода».) Разбейте сценарии использования на составные задачи. Каждая из них должна быть концептуально отдельной. Обратите особое внимание на начальное и конечное состояния задачи. Функции, созданные для этих задач, можно использовать в сценариях тестирования. При изменении пользовательского интерфейса задачи вам нужно будет обновить только саму задачу, а не тесты, которые ее используют. Это обеспечивает значительную степень абстракции, но при этом может потребоваться значительный объем работы по проектированию и созданию.
- **Автоматизация тестов на основе ключевых слов.** (См. урок 128 «Атоматизация тестов на основе ключевых слов позволяет непрограммистам легко создавать тесты».)
- **Автоматизация на основе API.** (См. урок 132 «Автоматизируйте тестирование через программные интерфейсы».) Полностью откажитесь от GUI.

Выбирайте средства тестирования GUI на основе совместимости, хорошего владения и обслуживания

Нас часто просят порекомендовать инструменты тестирования. Выбор оптимального варианта зависит от конкретной ситуации. Некоторые средства тестирования GUI несовместимы с конкретными средами разработки или плохо их поддерживают. Трудноуловимые факторы могут стать причиной проблем. Тяжело заранее предсказать, что будет работать, а что — нет. Это действительно вопрос проб и ошибок. Итак, с чего же начать?

Выясните, какими инструментами уже владеет ваша команда. Или знают ли они язык, используемый одним из инструментов. Затраты на подготовку и обучение работе с инструментом (будь то формальные занятия, самостоятельное изучение или метод проб и ошибок) зачастую весьма значительны. Хорошее владение инструментом может стать решающим фактором успеха в работе с ним, если, конечно, предположить, что он действительно работает.

Другой важный фактор — способность поставщика поддерживать инструмент. Даже если он хорошо работает с вашим продуктом сегодня, что произойдет, когда вам придется тестировать его на новой платформе в следующем году? Приобретая инструмент, вы инвестируете в способность поставщика сопровождать его и поддерживать в актуальном состоянии с помощью новых технологий. Посетите их форумы онлайн-поддержки и поговорите с другими пользователями об истории поставщика.

Вам потребуется некоторое время, чтобы проверить совместимость инструмента с вашим продуктом и изучить послужной список поставщика. Приобретите инструмент на пробной основе (30–90 дней) или по крайней мере получите 30-дневную гарантию возврата денег. За это время вы можете столкнуться с какими-либо проблемами. Проверьте, насколько своевременны и информативны ответы службы поддержки. Кроме того, вы должны попросить добавить обучение в пакет с покупкой инструмента. Таким образом вы можете приобрести его по более выгодной цене. Мы также обнаружили, что важные советы по использованию некоторых инструментов можно получить только в процессе обучения. В руководствах к ним этих советов нет.

У нас есть свое мнение по поводу того, какие функции полезны в том или ином инструменте, но для большинства команд вышеизложенные соображения позволяют свести выбор к единственному варианту. Дополнительную информацию см. в книге Хендриксона (Hendrickson, 1999).

Автоматизированные регрессионные тесты становятся бесполезными

Самая большая проблема, с которой сталкиваются автоматизированные регрессионные тесты, — это деградация и досрочный вывод из эксплуатации. Вы разрабатываете эти тесты для обнаружения проблем в уже протестированных функциях — проблем, возникающих из-за ошибок программистов, совершаемых при исправлении багов или добавлении новых функций.

Ниже представлены несколько примерных причин, которые приводят к деградации регрессионных тестов.

- **Изменение пользовательского интерфейса или форматов вывода.** Эти изменения — основная причина деградации. Из-за них большой набор тестов, которые раньше проходили успешно, терпит неудачу, даже если не было сделано ничего очевидного, что могло повлиять на функции, с которыми работают тесты. Изменения могут быть настолько незначительными, что специалисты по ручному регрессионному тестированию едва ли обратят на них внимание. Однако автоматизированные тесты чувствительны и хрупки, они не способны отличить улучшения от багов.
- **Заложенные при проектировании предположения о тестовой среде.** Наборы тестов могут выйти из строя при переносе на другие машины или при перемещении необходимых ресурсов.
- **Ошибки при сопровождении.** Автоматизаторы, исправляющие тесты, допускают ошибки, внося баги в наборы. Таким образом, в наборах регрессионных тестов сами собой возникают регрессии багов.
- **Смена операторов.** Работа с наборами тестов и их сопровождение могут потребовать специальных навыков и знаний. Когда автоматизатора тестов переводят на другую должность, ценные знания могут быть утрачены. Например, Шейла временно отключает тесты дискового ввода-вывода из-за конкретного бага в продукте, а затем уходит работать программистом в другую компанию. Заменивший

> Шейлу специалист считает, что тесты полностью устарели, и переносит их в архив, чтобы никогда больше их не видеть.

Тестировщики вкладываются в автоматизацию регрессий, рассчитывая на то, что эти инвестиции окупятся в свое время. К сожалению, они часто обнаруживают, что тесты перестают работать раньше, чем ожидалось. Тесты устаревают, перестают находить баги и нуждаются в исправлениях. Тестировщики адекватно реагируют на это, обновляя тесты. Но это может оказаться сложнее, чем ожидалось. Многие автоматизаторы тратят слишком много времени на диагностику отказов деградировавших тестов и их исправление.

Проявляйте осторожность при выборе тестов для автоматизации, поскольку все, что вы автоматизируете, вам предстоит сопровождать или придется от этого отказаться. Неконтролируемые эксплуатационные расходы, пожалуй, самая распространенная проблема, с которой приходится сталкиваться при работе с автоматизированными регрессионными тестами.

Уже только эксплуатационные расходы должны убедить вас не верить сказкам о том, что автоматизация превратит вашу работу в отдых.

Автоматизация тестирования — это процесс разработки программного обеспечения

Проекты по автоматизации тестирования часто терпят неудачу из-за отсутствия дисциплины и навыков управления проектом. Многие тестировщики не осознают, что, автоматизируя тесты, фактически разрабатывают программное обеспечение.

Вайнберг представляет шкалу оценки организационного развития (Weinberg, 1992). Как и в аналогичных шкалах, первый уровень — беспорядочная организация, второй — повторяемый и так далее до пятого уровня. Однако на шкале Вайнберга есть дополнительный уровень в самом низу. Нулевой уровень — это *незнающая* организация. Эти организации даже не подозревают о том, что разрабатывают программное обеспечение. Многие команды тестировщиков, не осознающие, что автоматизация тестирования — это разработка ПО, попадают в эту категорию.

Любой успешный проект по разработке программного обеспечения требует соблюдения некоторых основных правил. Тестировщики часто жалуются первыми, когда программисты не следуют правилам. Тестировщики

не должны удивляться, когда проекты по автоматизации тестирования сталкиваются с проблемами после того, как они сами нарушают правила.

Каковы правила? Спланируйте проект и определите этапы и результаты. Определите требования. Используйте систему управления версиями для хранения инструментов, кода автоматизации и тестов. Проектируйте тесты перед их реализацией. Проводите ревью кода и тестируйте его. Отслеживайте баги автоматизации в базе данных багов. Документируйте использование автоматизации и учтите, что она будет использоваться не только теми, кто ее создал.

Отказ рассматривать автоматизацию как программную инженерию привел к дорогостоящим фиаско с автоматизацией, например к тому, что команды тестировщиков оказались в числе аутсайдеров, не имея полезных автоматизированных тестов.

Мы не будем говорить вам, какому именно процессу разработки следовать, но делать это все же сто́ит.

Автоматизация тестирования подразумевает большие инвестиции

Автоматизация тестов требует времени и денег. Ожидайте, что на автоматизацию хорошо спроектированного автоматизированного теста с помощью инструмента GUI-тестирования потребуется в десять раз больше времени, чем на его ручное выполнение. Эта цифра зависит от множества факторов, но многие опытные автоматизаторы используют ее в качестве обоснованной первичной оценки (Kaner, 1998a). Мы рассмотрели несколько утверждений о том, что тесты можно автоматизировать, приложив всего лишь в два-три раза больше усилий. Мы считаем, что эти утверждения являются результатом разработки одноразовых тестов, неспособности учесть все усилия, затрачиваемые на автоматизацию, или удачи. Если вы благоразумны, то не станете использовать более низкую цифру, пока не получите конкретные доказательства, характерные для ваших условий.

Эффективное использование средств тестирования предполагает квалифицированное программирование и проектирование. Как и автоматизация любой другой деятельности, автоматизация тестирования — проблема программной инженерии.

Автоматизация тестирования может помочь сократить расходы, если у вас есть веские причины для многократного использования одного и того же теста. Она может помочь создать и выполнить тесты, которые дорого или невозможно выполнить вручную. С ее помощью можно измерить те аспекты работы программы (время и использование памяти), которые невозможно измерить вручную. Автоматизация позволяет провести обширную серию тестов в целях поиска дефектов, которые проявились бы только через несколько месяцев нормальной работы и обнаружение которых заняло бы слишком много времени при ручном выполнении. Любое из этих преимуществ может оправдать значительные инвестиции в автоматизацию тестирования, но если вы будете составлять штат и бюджет проекта автоматизации, исходя из предположения, что его можно выполнить на полставки почти без затрат, то вас ждет разочарование.

На момент публикации многие средства тестирования GUI стоят около 5000 долларов за одно рабочее место. Инструменты нагрузочного тестирования могут стоить 50 000 долларов и выше. Как бы ни были велики эти затраты, мы считаем, что их быстро превышают расходы на обучение персонала (формальное или на рабочем месте) и внедрение. Не совершайте ошибку, тратя весь бюджет, выделенный на автоматизацию тестирования, на инструменты тестирования. Это лишь вершина айсберга.

Проекты по автоматизации тестирования требуют навыков программирования, тестирования и управления проектами

Нечасто можно встретить людей, хорошо разбирающихся *как* в разработке тестов, *так* и в их автоматизации. Обычно важно поручать эти виды деятельности разным людям. И разработка, и автоматизация — это работа на полный рабочий день. Использование специалистов дает наилучшие результаты.

- **Тестирование.** Сформулируйте цели тестирования для автоматизации. Для чего вы будете проводить тесты? Как они помогут вам найти баги? Какие это будут баги? Основаны ли тесты на понимании предметной области продукта? Программисты с благими намерениями, не имеющие достаточного представления о тестировании, могут создавать замысловатые, но не несущие никакой ценности наборы тестов. Обеспечьте руководство со стороны людей, которые понимают, что такое тестирование и как будет использоваться продукт.

- **Программирование.** Автоматизация тестирования — это программирование. Стратегии инструментов, которые обещают позволить тестировщикам создавать тестовые наборы без программирования, не работают. Не стоит также полагаться только на начинающих и неопытных программистов. Управление, установка, настройка и сопровождение инструментов требуют навыков программирования. Каждый автоматизированный тест — это программа или функция в более крупном приложении автоматизированного тестирования. Автоматизация тестирования — дело непростое и не принесет успеха, если не будут соблюдаться принципы программной инженерии.
- **Управление проектом.** Без должного внимания со стороны руководства проекты автоматизации могут фактически не достичь первоначально поставленных целей. Не превращайте автоматизацию в побочный проект. Не нанимайте сотрудников, работающих неполный рабочий день.

Хороший баланс навыков особенно важен для проектов, в которых создаются тесты, предназначенные для использования в течение длительного времени. Вам понадобятся люди, обученные тем языкам и инструментам, которые задействуете вы. Подход к автоматизации, который вы используете, будет определять точное сочетание навыков.

Используйте пилотные проекты, чтобы доказать целесообразность

С помощью пилотного проекта автоматизации вы можете проверить правильность своего подхода, убедиться в том, что ваши инструменты работают с вашим продуктом, и определить, какую выгоду принесет автоматизация. Планируйте показать результаты через месяц или около того. Затем можно переходить к более масштабным мероприятиям.

Пилотный проект поможет продемонстрировать ваши возможности. Это позволит получить ресурсы и установить сотрудничество, которые необходимы для успешной реализации более масштабного проекта.

Автоматизированные тесты изменят ваши процессы разработки и тестирования. Чем раньше они у вас появятся, тем быстрее вы начнете вносить изменения. Чем дольше ваша команда использует ручные процессы тестирования, тем сложнее получить максимальную выгоду от применения автоматизированных наборов.

Поручите тестировщикам и программистам составить устав проектов автоматизации

Проекты по автоматизации выигрывают, если реализуются под совместным руководством тестировщиков и программистов.

- **Тестировщики.** Предоставьте им возможность определять требования к автоматизации тестирования. Кроме того, попросите их убедиться в том, что разработанная автоматизация полезна, понятна и заслуживает доверия. Недостаточно, чтобы автоматизация хорошо тестировала. Она также должна заслужить доверие тестировщиков. Если они не доверяют ей, то не будут ее использовать. Автоматизация должна служить интересам тестировщиков.
- **Программисты.** Программисты являются экспертами в области разработки ПО и должны проанализировать архитектуру автоматизации. Благодаря их привлечению улучшается сотрудничество и расширяются возможности тестирования продукта.

Поставьте четкие цели и определите требования к следующим ключевым областям, которые часто упускаются из виду (Pettichord, 1999).

- **Возможность проведения ревью.** Кто должен иметь возможность проводить ревью тестов? Насколько сложно это будет сделать?
- **Сопровождаемость.** Кто будет сопровождать тесты? Что эти люди должны будут знать?
- **Честность.** Как вы узнаете, что тестам можно доверять?

Проектируйте автоматизированные тесты так, чтобы их легко было проверить

Вы тестируете программное обеспечение, поскольку в коде случаются ошибки. Однако ошибки допускаются и в тестовом коде. И как поступать с ними?

Вы можете проверить свой тестовый код. Хорошая идея. Следует ли автоматизировать это тестирование? Нужно ли тестировать код, автоматизирующий тестирование вашей автоматизации? Даже просто мысли об этом причиняют боль.

Другой подход заключается в том, чтобы сосредоточиться на проведении ревью тестового кода. Проектируйте наборы тестов так, чтобы их могли

проверять разные люди. Это поможет им развить веру в тесты, которые пишет ваша команда, и в тесты, которые пишете вы. Сохраняйте простоту кода. Оптимизируйте форматы тестовых данных под нужды тестировщиков. Используйте стандартные языки программирования.

Автоматизация тестирования не застрахована от ошибок, но если сделать возможным проведение ревью, то вероятность серьезных ошибок значительно снизится. Кроме того, вы помогаете своей команде учиться друг у друга. Научившись проводить ревью тестового кода, они также смогут научиться проводить ревью и кода продукта.

Культура, поощряющая проведение ревью, приводит к автоматизации тестирования, на которую с полным основанием может положиться каждый.

Мы имеем ограниченный опыт парного программирования автоматизированных тестов. Это еще одна форма проверки, и мы ожидаем (но не можем этого утверждать, не имея опыта), что она будет работать хорошо.

Не экономьте на разработке автоматизированных тестов

Можно предположить, что достаточно умный и наблюдательный человек будет выполнять ручные тесты. В отношении автоматизированных тестов такого предположения быть не может.

При разработке автоматизированных тестов вам, возможно, придется:

- убедиться, что тест настроен правильно;
- указать ожидаемые результаты;
- обратить внимание на возможные ошибки и побочные эффекты;
- восстанавливать тесты после возможного отказа;
- предотвращать взаимное влияние тестов друг на друга.

Документируйте дизайн тестов, чтобы люди, которые будут использовать их в дальнейшем, имели представление о том, о чем вы думали.

Избегайте сложной логики в тестовых сценариях

Условная логика в таких сценариях осложняет понимание тестов и повышает вероятность ошибок. Еще больше проблем может доставить код, генерирующий и перехватывающий ошибки.

Вам может понадобиться логика для управления настройкой теста, реагирования на проверенный вывод или для обработки пользовательских элементов управления. Поместите эту логику в отдельные функции. Вы сможете тестировать их изолированно (что хорошо) и более легко проводить ревью ваших тестов (что тоже хорошо).

Благодаря сохранению линейности тестов вы сможете сосредоточиться на цели тестирования (еще один положительный момент), а не на поддержке автоматизации. Когда тесты становятся слишком сложными, они, как правило, вызывают баги. Сохраняйте простоту. Сохраняйте линейность тестов.

Не создавайте библиотеки тестов только для того, чтобы избежать повторения кода

Стандартная мудрость программирования рекомендует избегать повторяющегося кода, помещая его в функцию, которая вызывается из каждого места, где находился этот код. В автоматизации тестирования такой подход часто приводит к проблемам. Противоположный подход, при котором повторяющийся код оставляют как есть, называется *явным кодированием*.

По своей природе тесты содержат большое количество повторений. Вы тестируете одни и те же функции в разных сценариях, в разном порядке или в сочетании с другими функциями. Если просто перемещать повторяющийся код, то в итоге получится сборная библиотека. Функции в такой библиотеке могут содержать последовательности проверок, которые часто следуют друг за другом, даже если являются частью разных задач. Соглашения об именовании, анализ результатов и стратегия тестирования также могут оказаться частью сборной библиотеки. Функциям в этой библиотеке трудно будет подобрать осмысленные имена и трудно понять, что они делают.

Тесты, использующие такие библиотеки, трудно подвергать ревью, отлаживать и сопровождать. Мы неоднократно проводили анализ наборов тестов, созданных с помощью сборных библиотек. Результаты никогда не бывают хорошими.

Автоматизация тестирования подразумевает использование большого количества повторяющегося кода. Чтобы создать полезные библиотеки, нужно следовать более жестким принципам проектирования, чем просто избегать повторения кода. Полезные библиотеки задач предназначены

для инкапсуляции задач, воспринимаемых пользователем, при этом особое внимание уделяется начальному и конечному состоянию функций. Эти усилия не всегда оправданны. В таких случаях следует придерживаться явного кодирования.

Автоматизация тестирования на основе данных упрощает запуск множества вариантов теста

Для тестирования различных входных данных и их сочетаний с помощью общей процедуры тестирования используйте автоматизацию тестирования на основе данных.

Занесите входные и ожидаемые результаты тестирования в таблицы, где каждая строка представляет собой тест. Затем создайте автоматизированную процедуру тестирования, которая считывает строку из таблицы, выполняет каждый шаг ввода и проверяет ожидаемый результат. В электронных таблицах удобно хранить тестовые данные. Таблицы же облегчают ввод данных. Без особых проблем получить доступ к данным электронных таблиц могут большинство инструментов тестирования и сред программирования. Они могут получать доступ к данным в собственном формате электронных таблиц или в виде текстовых файлов с разделителями, которые можно легко экспортировать (CSV-файлы).

Создав процедуру тестирования на основе данных, можно использовать ее снова и снова для выполнения новых тестов. Эта методика наиболее эффективна при работе с продуктами, имеющими множество различных вариантов данных. Для поддержки тестов, состоящих из различных последовательностей или нескольких альтернативных путей, можно использовать более сложный вариант — *автоматизацию на основе ключевых слов*.

Автоматизация тестирования на основе данных поддерживает тестировщиков, не занимающихся программированием. Автоматизаторы создают процедуры тестирования на основе данных, а тестировщики — тестовые данные. В некоторых случаях бывает сложно автоматизировать проверку результатов теста. Чтобы упростить ручной анализ результатов, сделайте так, чтобы тестовая процедура собирала результаты и представляла их в контексте входных данных.

Автоматизация тестирования на основе данных становится обычным явлением. В многие инструменты тестирования встроена прямая поддержка этого метода (Dwyer, Freeburn, 1999).

Автоматизация тестирования на основе ключевых слов позволяет непрограммистам легко создавать тесты

Автоматизация тестирования на основе ключевых слов опирается на метод на основе данных. Однако в таблицах содержатся не просто данные, а директивы (ключевые слова).

Во-первых, для реализации этого подхода необходим общий фреймворк, предоставляющий поддержку выполнения тестов, а также библиотеки для настройки, анализа результатов и создания отчетов. Все это будет использоваться для всех тестов на основе ключевых слов.

Во-вторых, необходимо создать библиотеку задач, в которой будут собраны все пользовательские задачи, поддерживаемые вашим продуктом. Определите каждую функцию задачи, которая может быть использована при тестировании, и добавьте для каждой из них запись в библиотеку задач. Объявите начальные состояния, в которых функции задачи допустимы, и конечные состояния, к которым они приводят. Это позволит определить, какие последовательности функций задачи являются допустимыми, и таким образом отсечь некорректные тесты.

В-третьих, добавьте поддержку чтения данных из электронных таблиц по одной строке. Используя объявления, интерпретируйте первый столбец как имя функции задачи. Следующие столбцы — как аргументы функции. Выполните функцию с ее аргументами. Затем перейдите к следующей строке.

Результатом всех этих действий становится *автоматизация тестирования на основе ключевых слов*. Ее использование позволяет *избежать сложной логики в тестовых сценариях* (см. урок 125). Тесты хранятся в электронных таблицах, поэтому непрограммистам часто легко их создавать и проводить их ревью. Поскольку все задачи, которые необходимо использовать и протестировать, задаются тестировщиком, он может сосредоточиться на тестах, а не на языке управления.

В условиях, когда создание автоматизированных тестов требует участия многих непрограммистов, мы считаем, что это одно из лучших решений. Единственным недостатком такого подхода является то, что вы не сможете написать тест, если для требуемых задач уже существуют поддерживаемые ключевые слова. Определение и реализация функций задачи может потребовать больших усилий.

Рецензент Ханс Бувальда сообщает: «Подходы, основанные на ключевых словах, как и мои собственные слова-действия, могут послужить хорошей опорой. Однако с годами я понял, что тестирование и его автоматизация остаются очень сложными областями и их правильное выполнение требует участия опытных специалистов».

Мы наблюдали хорошие результаты применения этого метода в проектах, в которых на разработку автоматизации отводилось значительное время. Но у нас есть и отчеты команд, которые обнаружили, что такой подход требует слишком больших накладных расходов и является неприемлемым.

Подробнее об этом методе рассказывают в своих работах Петтикорд (Pettichord, 1996), Бувальда и Касдорп (Buwalda, Kasdorp, 1999).

Автоматизируйте генерирование входных данных для тестов

Общие методы программирования могут помочь в нескольких ситуациях.

- **Создание больших файлов.**
- **Создание большого количества входных данных для тестов.**
- **Настройка тестовых окружений.** При нагрузочном тестировании начните с предварительной загрузки в базу данных реалистичного объема данных. Объем данных, доступных для поиска, влияет на извлечение информации из базы.
- **Создание случайных данных.** Это особенно полезно в случае тестов на основе данных и ключевых слов.
- **Покрытие всех сочетаний входных данных.** Используйте алгоритмы для создания перестановок и комбинаций.

Однако при использовании предыдущих методов ожидаемые результаты обычно остаются неопределенными. Это может повлечь за собой трудоемкую работу по проверке или уточнению результатов вручную. Указанные ниже методы особенно ценны тем, что позволяют либо сократить количество необходимых тестовых случаев, либо обеспечить определенный уровень покрытия, либо уточнить ожидаемые результаты.

- **Покрытие всех пар представителей классов эквивалентности.** Мы встречали исследования, в которых утверждалось, что большинство багов взаимодействия можно найти, если протестировать все

парные сочетания ключевых членов класса эквивалентности. Техника попарного тестирования рассматривается в подразделе «Как проводить комбинированное тестирование с помощью техники попарного тестирования» в главе 3.

- **Покрытие взаимодействий между логическими условиями.** Когда переменные не являются независимыми, такие техники, как комбинированное попарное тестирование, не работают. Более надежным подходом является построение *графов причинно-следственных связей* (Elmendorf, 1973; Bender, 1991). Мы не использовали эту технику, но слышали и о неудачных, и об успешных попытках ее применения.
- **Создание сценариев тестирования с помощью моделей состояний.** Техника тестирования, в которой используются модели состояний, подверглась серьезному академическому исследованию, и были получены значимые результаты в промышленности. Модель состояний, или диаграмма состояний, определяет состояния системы (документ изменен или не изменен; база данных подключена или отключена; транзакция ожидает или завершена) и возможные переходы между ними. Одни специалисты, имеющие много опыта, способны построить интересные и полезные модели, не создавая слишком много состояний. Другие же тратят на эту технику много усилий, создают огромные модели состояний и не получают никакой пользы. Это называется проблемой *взрыва состояний*. Те специалисты, которые добиваются наибольшего успеха, часто видят результаты в короткие сроки. Если вы используете эту технику, то мы советуем вам создать модель состояний для одной-двух функций, создать тесты, а затем с ее помощью провести повторную оценку. Если недельные усилия по созданию модели не окупаются, то, скорее всего, не стоит продолжать в нее вкладываться (Robinson, 1999; Nyman, 2000).

Отделите создание теста от его выполнения

Одна из стратегий, позволяющих отделить тестовые данные от выполняемого кода, — автоматизация тестирования на основе данных (см. урок 127). Такое разделение облегчает создание тестов и имеет ряд преимуществ.

- Ваши тесты будет легче понять и проверить.
- Для создания и выполнения можно использовать различные тестовые инструменты или среды программирования.

- Отдельный генератор тестовых случаев легче тестировать. Если вы используете случайные методы, то должны знать, что алгоритмы случайных чисел, поставляемые со средами программирования, часто оказываются слабыми. Ваши данные могут оказаться менее случайными, чем вы ожидали. Проверьте и убедитесь в этом (Park, Miller, 1988). Кейнер и Воки (Kaner, Vokey, 1984) предлагают тщательно протестированный набор параметров для генератора случайных чисел, который можно легко запрограммировать на Java или любом другом языке, работающем с целочисленной арифметикой высокой точности.
- Тесты легче повторять, когда данные сгенерированы предварительно. Мы видели сценарии тестирования, которые меняли тест при каждом запуске. Если нет возможности сгенерировать данные предварительно, то необходимо принять другие меры для обеспечения повторяемости, например протоколировать данные или начальное число, использованное для их генерации. Оснащение сценариев тестирования инструментами, позволяющими использовать эти журналы, усложняет задачу.
- Вам будет легче сообщать о любых обнаруженных багах. Программист может сомневаться в ваших инструментах.
- Разные специалисты по тестированию могут заниматься различными аспектами автоматизированного тестирования, используя те инструменты и языки, которые они считают наиболее подходящими.

Используйте стандартные скриптовые языки

Если вы — тестировщик, который хотел бы больше узнать о программировании, то мы советуем вам изучить Perl, Visual Basic, TCL, JavaScript, Python или любой другой скриптовый язык, который знают и используют программисты из вашего окружения (Sweeney, 2001). Некоторые скриптовые языки, такие как скрипты оболочки Unix или пакетные файлы DOS, существуют уже давно. Скриптовые языки — это языки высокого уровня, оптимизированные больше для удобства использования, чем для повышения производительности.

Многие программисты работают более продуктивно и совершают меньше ошибок, когда используют скриптовые языки вместо системных языков программирования, таких как C/C++ или Java.

Скриптовые языки очень хорошо подходят для автоматизации большинства тестов. С их помощью можно генерировать тестовые данные, получать доступ к интерфейсам программирования и проверять результаты.

Многие инструменты тестирования имеют встроенные скриптовые языки. Одни используют стандартные языки, другие — узкоспециализированные, которые мы называем *языками производителя*. Мы не видим веских причин для их использования и отмечаем несколько проблем.

- **Они затрудняют кодирование.** Многие из них основаны на стандартных языках, таких как C. Зная C, вы сможете прочитать сценарий на *языке производителя*, но потратите много времени на написание правильного кода. Эти сценарии не поддерживают многие стандартные языковые идиомы.
- **Их трудно изучать.** Курсы или книги, посвященные таким языкам, редки. А если вы освоите такие сценарии, то не сможете использовать их ни для чего другого. Поэтому трудно мотивировать людей изучать эти языки. А при найме сотрудников вам будет сложно найти людей, которые уже знают их.
- **Они мешают совместной работе тестировщиков и программистов.** Мы рекомендуем сделать так, чтобы в рамках проектов автоматизации сотрудничали как тестировщики, так и программисты продукта. Это становится сложным, когда вы буквально используете разные языки.
- **Трудно опираться на чужой труд.** Библиотеки, доступные для языков производителя, имеют мизерный размер по сравнению с теми, которые доступны для стандартных языков. Это означает, что вы не сможете использовать наработки других людей, а будете тратить свое время на переустройство примитивных библиотек.

В настоящее время все больше средств тестирования используют стандартные языки. Мы рекомендуем избегать инструментов, в которых используются языки производителя. Если вы вынуждены работать с таким инструментом, то постарайтесь уменьшить объем кода, который должен быть на нем написан, и выполняйте как можно больше операций в отдельной языковой среде[1].

---

[1] Дополнительную информацию см. в книге Петтикорда (Pettichord, 2001a).

Автоматизируйте тестирование через программные интерфейсы

В настоящее время многие продукты имеют программные интерфейсы (публичные API), которые можно использовать для тестирования. Те продукты, у которых их нет, могут иметь скрытые интерфейсы (частные API), к которым вы можете получить доступ, если попросите об этом.

Публичные API документируются как часть программного продукта. Они не могут сильно измениться и благодаря своей стабильности подходят для автоматизации тестирования. Частный API может быть очень полезен, но необходимо выяснить, насколько велика вероятность его изменения.

Не зная об этих альтернативах, многие автоматизаторы сосредотачиваются на том, что наиболее заметно, — на GUI. К сожалению, обычно этот интерфейс сложнее всего автоматизировать. Практически с любым интерфейсом программирования легче работать — он более стабилен и надежен[1]. В предыдущих уроках мы обсудили некоторые проблемы автоматизации GUI и предлагали стратегии их решения. Однако лучший подход — вообще избегать их. Интерфейсы программирования, как правило, обеспечивают стабильность. Они также облегчают обнаружение и изоляцию ошибок.

Анализируя многочисленные примеры тестирования продуктов, мы пришли к выводу, что существует тесная взаимосвязь между наличием интерфейсов программирования для тестирования и разработкой

---

[1] Пол Шимковяк (Paul Szymkowiak) с этим не согласен: «Мой опыт другой. Я обнаружил, что многие пользовательские интерфейсы более стабильны, чем доступные интерфейсы программирования, особенно применительно к целям автоматизации тестирования. По-видимому, это связано с тем, что пользовательский интерфейс виден заказчикам, используется в учебных материалах и описывается в руководствах пользователя. В связи с этим больше затрат связано с постоянным внесением в него изменений. Я работал с несколькими руководителями проекта, которые требовали заморозить пользовательский интерфейс задолго до того, как был заморожен эквивалентный интерфейс программирования. Многие интерфейсы программирования плохо документированы, и, поскольку целевыми потребителями являются программисты, поставщик/разработчик ПО полагает, что они не будут возражать против постоянных изменений интерфейса. Я также обнаружил, что многие интерфейсы программирования полны багов и имеют явно более низкий “порог удобства использования”, чем обычно ожидается от пользовательского интерфейса».

мощных автоматизированных тестовых наборов[1]. К интерфейсам программирования относятся API, интерфейсы командной строки, COM-интерфейсы, HTTP и др. Вам придется изучить язык и технологию. Не ждите, что ваши программисты дадут вам учебник. Мало у кого хватит терпения и времени. Если вы серьезно относитесь к автоматизации, то научи́тесь или попросите кого-то, кто в ней разбирается, помочь вам.

Даже автоматизаторы GUI-тестов сталкиваются с необходимостью изучения деталей технологии GUI. На техническом уровне она действительно сложнее, чем другие интерфейсные технологии. Так или иначе, вам придется изучать детали технологии интерфейса, которую используют ваши автоматизированные тесты. Выбор за вами. Мы считаем, что GUI должен быть последним в вашем списке.

Приведем небольшой пример. Многие люди спрашивают у нас совета по автоматизации инсталляторов с помощью GUI-инструментов тестирования. Как правило, лучший подход заключается в том, чтобы отказаться от инструмента. Большинство инсталляторов имеют интерфейсы сценариев, обеспечивающие более хороший подход. Например, InstallShield — популярная система установки, которая используется для создания инсталляторов многих продуктов. Многие тестировщики не знают, что можно запускать программы установки InstallShield с возможностью записи выбранных параметров установки. Они сохраняются в файле ответов. Впоследствии программу установки можно запустить без посторонней помощи, используя файл ответа для указания параметров установки. Это просто и дешево, а файл ответов легко читать и редактировать. Это недорогой и удобный способ автоматизации установки (см. раздел «Создание автоматической установки» на http://support.installshield.com/kb/display.asp?documents_id=101901).

Но вы не тестируете графический интерфейс! Сосредоточьтесь на том, где автоматизация может быть наиболее полезной. Иногда можно

---

[1] Дуглас Хоффман имеет схожий опыт: «Наиболее значительного успеха я добился, работая с известным пакетом для настольных издательских систем. Его движок был незаметен, и продукт воспринимался с точки зрения его графического интерфейса WYSIWYG. В выпуске n + 1 планировалось полностью переделать пользовательский интерфейс, поэтому мы не стали автоматизировать тестирование GUI. Вместо этого мы использовали общедоступный API для автоматизации функциональных тестов и вручную тестировали GUI. Мы выявили множество дефектов (в основном обнаруженных при автоматизации тестов) и смогли гораздо легче отделить проблемы GUI от функциональных проблем».

эффективно автоматизировать GUI, иногда — нет. Не позволяйте людским предрассудкам относительно того, как должна выглядеть автоматизация, ограничить вас.

Поощряйте разработку наборов модульных тестов

Модульное тестирование фокусируется на мельчайших единицах, составляющих программную систему: функциях, классах и методах, создаваемых программистами. Большинство руководителей ожидают, что их программисты будут проводить модульное тестирование, и большинство программистов утверждают, что они это делают. Однако реальная практика чрезвычайно разнообразна, и зачастую ее трудно подтвердить.

При настоящем модульном тестировании блоки тестируются изолированно. Для обработки исходящих вызовов создаются заглушки, а для имитации входящих вызовов — драйверы. Создание таких заглушек может потребовать значительных усилий.

Более популярная форма автоматизированного модульного тестирования позволяет избежать разработки заглушек. Для этого модули тестируются в контексте. Мы можем назвать это *интеграционным модульным тестированием*. Для систем, которые строятся снизу вверх, такая форма автоматизации может быть достаточно простой.

Управлять выполнением набора тестов можно с помощью фреймворков, таких как Junit или Xunit. Это не слишком сложно и не слишком дорого. Тестирование кода осуществляется через привычный интерфейс вызова функций, поддерживаемый языками программирования. Программисты пишут модульные тесты на том же языке, что и программное обеспечение продукта: тесты для языка Java — на Java, тесты для языка C — на C. Используйте модульные тесты для регрессионного тестирования, смоук-тестирования и тестирования конфигурации.

Мы рекомендуем воздерживаться от указаний программистам, что им делать. Но если руководители требуют большей автоматизации тестирования, то должны знать, чем могут помочь программисты и тестировщики. Если программисты проявляют интерес к модульному тестированию, то мы предлагаем помочь им. Модульное тестирование рассматривается как основная практика экстремального программирования и других гибких методов (Beck, 1999; Beck et al., 2001).

Остерегайтесь привлекать к работе автоматизаторов, которые не разбираются в тестировании

Для автоматизации нужны программисты. Какие программисты вам нужны?

Большинство программистов считают, что много знают о тестировании. Если они не крайне безответственны, то часто тестируют собственный код и нередко обнаруживают, что им приходится тестировать другой код, который они пытаются использовать. Они действительно много знают о тестировании, но только не по сравнению с тем, кто целиком посвящает себя тестированию: с таким человеком, как вы!

С точки зрения тестировщика, многие программисты на самом деле мало знают о тестировании ПО (так же, как и многие тестировщики мало знают о программировании). Знания программистов часто основаны на тестировании их собственного кода. Они переоценивают стратегии, соответствующие их личным особенностям. Люди, которые лучше понимают требования к тестированию, должны вносить свой вклад и проводить ревью.

Недостаток знаний о тестировании у программистов становится еще более серьезной проблемой в сочетании с пренебрежением, которое многие программисты испытывают к тестировщикам. Такое отношение обусловлено тем, что тестировщики недостаточно осведомлены о ПО в целом. Как следствие, программисты могут предоставить автоматизацию, которая явно отличается от того, что было запрошено или ожидалось. Чтобы избежать подобных ситуаций, планируйте ревью кода автоматизации тестирования. Программисты привыкли к тому, что им предъявляют требования, которые они не совсем понимают. Их стратегия обычно такова: они делают предположение, пишут программу, а затем исправляют ее на основе обратной связи. Такая стратегия вполне объяснима, учитывая ту неясность и путаницу, которые часто присущи требованиям и другим запросам к программированию. Опасность возникает, когда эта стратегия применяется к автоматизации тестирования. Не заблуждайтесь, полагая, что простой запуск тестового набора позволит определить, работает ли он. Одни баги в тестовом наборе приведут к тому, что тесты не будут работать; вы найдете их и исправите. Из-за других багов тесты будут успешными, хотя не должны быть таковыми, а вы об этом даже не узнаете. Чтобы избежать подобных ситуаций, проводите ревью, используйте стратегии проектирования и тестирование.

Некоторые ответственные программисты в процессе работы над проектом автоматизации могут понять, что знают о тестировании недостаточно. Они могут попытаться замедлиться, давая себе шанс научиться. Вместо этого предложите им продолжать работать в прежнем темпе, но над пилотными проектами и ревью, проводимыми тестировщиками. Или привлеките их к тестированию, чтобы они могли на собственном опыте узнать, что нужно тестировщикам.

Избегайте автоматизаторов, которые не уважают тестирование

Некоторые программисты считают тестирование низкоквалифицированной работой, не соответствующей их способностям. Мы не раз сталкивались с тем, что таких программистов назначали на должность тестировщика. Они, естественно, тяготеют к автоматизации тестирования, в которой, по их убеждению, преуспеют, поскольку считают себя блестящими программистами.

Считая тестирование чем-то тяжелым, такие люди не имеют стимула развивать его навыки. Вместо этого они требуют сделать работу интересной, зацикливаются на задачах автоматизации и начинают разрабатывать инструменты, ценность которых незначительна. Или переделывают систему отслеживания багов либо создают причудливый GUI для запуска тестов — занимаются чем угодно, лишь бы не тестировать.

Мы не видели, чтобы привлечение подобных специалистов пошло на пользу, не знаем, как правильно использовать их навыки, и советуем по возможности избегать таких людей. А если это невозможно, то убедитесь, что они не создадут проблем остальным членам команды тестировщиков. Возможно, вы сможете временно перевести их в службу технической поддержки. (И не забудьте похвалиться, что оказали ей такую большую услугу.)

Тестируемость часто является более выгодной инвестицией, чем автоматизация

Во многих случаях для поддержки тестов можно использовать либо тестируемость (код поддержки тестов *внутри* продукта, обеспечивающий контроль или видимость), либо автоматизацию (код поддержки тестов *вне* продукта). Благодаря тестируемости часто можно получить

лучшее решение, затратив меньше усилий. Приведем несколько примеров.

- После установки продукта пользователи (и тестировщики) должны были просмотреть журнал на предмет наличия ошибок установки. Как автоматизировать это тестирование? Первой идеей было написать скрипт, который бы искал в журнале возможные сообщения об ошибках. Более удачной идеей было встроить эту функцию в продукт. Она была бы более надежной, лучше бы тестировалась и принесла бы реальную пользу пользователям.
- Тестировщикам необходимо было смоделировать ошибки носителя в программном обеспечении для резервного копирования на магнитную ленту, чтобы проверить, корректно ли оно восстанавливается. Такие ошибки трудно смоделировать с помощью автоматизации. Вероятно, для этого потребовалось бы создать симулятор ленточного накопителя. Вместо этого тестировщики совместно с программистами сделали так, чтобы низкоуровневый код записи на ленту (ложный) мог указывать на неисправность носителя.
- Утверждения — это операторы в коде, сигнализирующие об ошибках, если предположения неверны. Утверждения можно поместить в тестируемое программное обеспечение, чтобы проверить обоснованность результатов. Зачастую это проще и эффективнее, чем писать внешний код для проверки результатов.

Тестируемость подразумевает наблюдение и контроль

Любые средства и возможности, помогающие наблюдать или контролировать работу программного обеспечения, улучшают тестируемость. Люди часто просят перечислить их. Что ж, сделаем это.

- **Доступ к исходному коду.** Многие компании не разрешают тестировщикам изменять исходный код. Однако доступ к просмотру кода все же может быть предоставлен. Особенно важно иметь возможность просматривать записи об изменениях в системе контроля исходного кода.
- **Журналирование.** Записывайте в журнал сообщения об ошибках, источники ошибок, профили использования, загрузку ресурсов, системные сообщения и протокольные взаимодействия. Предусмотрите возможность, позволяющую задавать различные уровни ведения журнала. Механизмы журналирования могут уже присутствовать в компонентах, используемых в вашем продукте. Программисты

используют журналы для отладки. С их помощью можно быстрее выявлять баги, анализировать их закономерности, представлять подробную информацию в отчетах, оценивать покрытие тестами, собирать информацию о работе с клиентами и узнавать больше о тестируемом программном обеспечении (DiMaggio, 2000; Johnson, 2001; Marick, 2000).

- **Диагностика.** Позволяет предупредить о потенциальных проблемах. Одним из примеров являются утверждения. Проверка целостности данных позволяет убедиться в их внутренней непротиворечивости. Проверка целостности кода помогает отследить, не были ли программы перезаписаны или изменены. Проверка целостности памяти дает возможность убедиться в том, что память используется в соответствии с ее распределением. В сочетании с журналированием диагностика может стать мощным инструментом, позволяющим при обнаружении ошибки перейти в режим отладки или сбросить информацию о программе (Kaner, 2000b)[1].
- **Имитация ошибок.** Ваш продукт имеет внутренние состояния, многие из которых, вероятно, трудно вызвать, особенно в воспроизводимой и систематической форме. Примером этого являются многие состояния ошибки. Ваше программное обеспечение должно уметь восстанавливаться после ошибок носителя информации, недостатка памяти или накопителя, задержек и разрывов в сети и других подобных проблем. Вы можете обнаружить, что эти состояния трудно создать, особенно воспроизводимым и систематическим образом. Чтобы облегчить тестирование обработки ошибок, триггеры состояния ошибки можно разместить на низком уровне ПО продукта. (Существуют также средства, позволяющие имитировать эти ошибки вне продукта (Houlihan, 2001).)
- **Тестовые точки.** Позволяют проверять или изменять данные в различных точках системы (Cohen, 2000).
- **Триггеры событий.** Уведомления о начале и окончании выполнения внутренних задач могут помочь синхронизировать тесты.
- **Чтение устаревших форматов данных.** В процессе разработки продукта форматы могут меняться несколько раз. Обеспечьте на-

[1] Ноэль Найман сообщает: «Есть два диагностических “инструмента”, которые Windows использует для поиска ошибок памяти. 1. Заполнять буферы известными шаблонами при их инициализации. Это помогает выявить выход за пределы памяти, неверные указатели и т. д. 2. Поместить буферы в конец выделенной кучи и работать в обратном направлении. Это приводит к тому, что переполнение буфера и другие подобные события вызывают уведомления Windows об ошибках памяти».

личие средств преобразования данных в новые форматы, чтобы не создавать их заново.

- **Тестовые интерфейсы.** Интерфейсы программирования — важное преимущество, позволяющее обеспечить тестируемость. В некоторых продуктах они добавлены именно по этой причине.
- **Поддержка пользовательских элементов управления.** Предоставление инструментам тестирования GUI возможности работать с пользовательскими элементами управления — это, пожалуй, одна из наиболее востребованных функций тестирования.
- **Поддержка запуска нескольких экземпляров.** Разрешите нескольким клиентам или агентам работать на одной машине, даже если это не поддерживается в реальных условиях. Это может позволить тестировщикам имитировать большую сеть в маленькой лаборатории.

Во многие программные продукты программисты добавили недокументированные возможности тестирования, чтобы облегчить себе работу по тестированию и отладке. Спросите о них[1].

Начинайте автоматизацию тестирования как можно раньше

Автоматизация — тяжелая работа, требующая планирования, исследований и проектирования. Если вы планируете проводить большое количество автоматизированных тестов, то начинайте работу еще на стадии разработки продукта. Эффективная автоматизация часто требует решения проблем тестируемости. Их трудно определить заранее, поэтому решения, связанные с тестируемостью, часто появляются в результате проб и ошибок.

Программисты более открыты для предложений по тестируемости на ранних этапах проекта. Функции легче добавлять, когда он еще не завершен. Программист и руководитель проекта могут добавить их в график и план и выделить для них бюджет. Неопределенность общего проектного решения побуждает многих программистов участвовать в тестировании; на программистов оказывается меньшее давление, если они знают, что код будет хорошо протестирован.

---

[1] В одном из проектов по разработке встраиваемого ПО Кейнер и Хоффман обнаружили, что программисты написали более 1100 диагностических команд для проверки состояния ПО и устройства. Все эти команды были доступны тестировщикам. Им нужно было только встроить соответствующие команды в свой инструмент выполнения тестов.

Некоторые менеджеры по тестированию не имеют такой возможности, но надеются, что смогут наверстать упущенное позже. Мы обнаружили, что чем позже начинается автоматизация, тем сложнее она проходит. Мы видим несколько причин этому.

- Когда тестирование идет полным ходом, трудно переключить ресурсы на автоматизацию.
- Когда персонал и процессы тестирования сосредотачиваются на ручном тестировании, они сопротивляются изменениям.
- Когда проект создан, программисты становятся менее сговорчивыми в отношении запросов на тестируемость.

Если вы хотите, чтобы автоматизация была успешной, то не откладывайте. Однако мы призываем вас не пытаться автоматизировать все с самого начала. Создавайте инфраструктуру на ранних этапах разработки, но подходите к выбору тестов для автоматизации с умом.

Предоставьте централизованным командам автоматизации четкие уставы

Некоторые компании создают централизованную команду автоматизации, которая поддерживает несколько команд по тестированию продуктов. Это часто оказывается разумным решением.

Если вы работаете в такой команде, то убедитесь, что у вас есть четкий устав, в котором описывается, какого рода помощь вы оказываете, как следует подавать запросы и как вы будете находить баланс в случае противоречивых требований. Это очень важно, поскольку команды, которые испытывают самые большие трудности и требуют наибольшей помощи, зачастую меньше всего способны извлечь выгоду из автоматизации. Участникам проще сказать, что им нужна помощь с автоматизацией (которая является техническим навыком), чем то, что им нужна помощь с организацией работы (что, как предполагается, они должны уметь делать сами). Если вы не справитесь с этим должным образом, то можете погрязнуть в неэффективности.

Мы предлагаем требовать от команд, получающих помощь, выделения сотрудников для совместной работы над проектами автоматизации. Это позволит получить ряд преимуществ.

- Вы можете оценить их заинтересованность и актуальные проблемы.

- Обучив персонал, можно снизить постоянные требования к сопровождению и анализу отказов.
- Взаимодействуя с теми, кто обращается к вам за услугами, вы с самого начала и на протяжении всего проекта делаете их требования к тестированию частью вашей работы.

Являясь централизованной командой автоматизации, вы представляете собой небольшую штатную команду программистов, имеющую прямой доступ к людям, которые обращаются к вам за услугами. Прочитайте книги Бека (Beck, 1999) и Джеффриса (Jeffries et al., 2000).

Автоматизация в целях немедленного воздействия

Слишком много людей считают, что автоматизация тестирования означает автоматизацию ручного тестирования. В результате чрезмерное внимание уделяется регрессионным тестам графического пользовательского интерфейса.

Вместо того чтобы думать о тестовых случаях, выберите легко достижимые цели, которые позволят повысить эффективность. Сосредоточьтесь на задачах, которые можно автоматизировать малыми усилиями и в результате получить высокую отдачу. Вот несколько хороших вариантов для начала.

- **Настройка и подготовка системы.** Новые тестировщики часто удивляются тому, сколько времени они тратят на установку и настройку систем, готовя их к тестированию. Эту работу можно автоматизировать. Образы дисков позволяют вернуть систему к стандартным конфигурациям. Автоматизацию настройки можно выполнять с помощью сценариев установки. Образцы данных можно загружать автоматически, используя обычные средства программирования.
- **Диагностическое обслуживание.** Некоторые ошибки трудно выявить. Дефекты, приводящие к повреждению данных или утечке памяти, обычно не обнаруживаются до тех пор, пока не будет осуществлен доступ к данным или не закончится память. Средства диагностики позволяют обнаружить эти сбои в момент их возникновения. Инструменты для исследования памяти доступны, и создание средств для проверки целостности данных или использования памяти в вашем продукте может оказаться несложным. Работайте с программистами проекта. Возможно, они уже используют такие инструменты.

- **Запись сеансов.** Тщательное составление баг-репортов требует внесения полной информации о конфигурации. Программы могут собирать и сообщать необходимую информацию автоматически. Эта возможность также позволяет помочь тестировщикам убедиться в том, что их система настроена должным образом.
- **Генерация тестовых данных.** *Используйте автоматизированные методы для создания входных данных теста* (см. урок 129).

Не обязательно автоматизировать тест от начала до конца. В первую очередь создайте некоторые автоматизированные средства. Это поможет перейти к более комплексным решениям.

У вас может быть больше инструментов тестирования, чем вы думаете

Для примера возьмем секундомер. Это отличный инструмент. Измерение времени отклика системы — один из важных видов тестирования. Секундомеры просты в использовании, адаптивны и точны. Некоторые тестировщики и программисты могут подумать, что автоматизированный подход, использующий вызов системных часов, обязательно будет лучше, однако измерять время в «черном ящике» с помощью секундомеров зачастую куда более удобно. Мы рекомендуем держать один из них под рукой.

Инструмент тестирования не обязательно должен иметь маркировку, сообщающую, что он является таковым. Тестировщики находят десятки других полезных инструментов. Многие из них дешевы и обычны. Назовем некоторые из них.

- **Средства создания образов дисков.** Позволяют быстро восстановить систему до известного состояния.
- **Проводники зависимостей.** Отображают динамические библиотеки, используемые программным приложением.
- **Сканеры файлов.** Обнаруживают и каталогизируют файлы, которые были изменены в системе.
- **Мониторы памяти.** Отслеживают использование памяти.
- **Макроопределения.** Упрощают повторение определенных задач.
- **«Маленькие языки», такие как Sed, Awk, Grep и Diff.** Эти инструменты упрощают автоматическое редактирование файлов, обработку выходных данных, поиск по данным и нахождение различий.

Первоначально разработанные для Unix, теперь они доступны на большинстве платформ.

Многие системные утилиты и средства общего программирования также отвечают заданным целям. Более того, зачастую они имеют достаточно низкую цену.

В Интернете можно найти много полезного бесплатного, условно бесплатного и дешевого ПО. Несколько полезных программ доступны на сайтах www.zdnet.com, www.pcmag.com, www.cnet.com, www.qadownloads.com и www.softpanorama.org. Элизабет Хендриксон собирает полезный набор ссылок на инструменты на сайте www.bughunting.com (см. также Hendrickson, 2001b).

Глава 6

Документирование тестирования

Мы написали эту главу для того, чтобы помочь вам изучить требования к тестовой документации. Мы не приводим образцы документов. (Примеры таблиц и матриц см. в главе 3.) Вместо этого мы приводим вопросы, которые могут помочь вам решить, что вам нужно.

Эта глава начинается с подробной оценки стандарта IEEE 829 «Документация по тестированию программного обеспечения». Мы понимаем, что вы, возможно, никогда его не читали и даже не слышали о нем, — IEEE продает его по достаточно высокой цене, которая, вероятно, отбивает желание у большинства людей покупать его копию. Мы встречали очень мало людей, у которых есть собственный экземпляр стандарта 829. Вообще, стандарты IEEE покупали немногие из наших клиентов или работодателей. Тем не менее многие шаблоны документации тестирования, распространенные в этой области, заимствованы из стандарта 829. Поэтому, даже если вы не знаете его по названию, если вы работаете в этой сфере уже какое-то время, то наверняка сталкивались с ним.

Если вы еще не имели дела со стандартом 829 и не интересуетесь шаблонами документации тестирования, то мы советуем вам перейти к уроку 147.

Документом, развивающим стандарт 829, является «Свод знаний по программной инженерии», который мы рассматриваем здесь.

В предисловии мы привели цитату из «Свода знаний по программной инженерии» (SWEBOK). Подробнее об этом будет сказано в главе 10. В текущей главе мы отмечаем позицию SWEBOK в отношении тестовой документации.

Тестовая документация и рабочие продукты

Документация — неотъемлемая часть формализации процесса тестирования. В стандарте IEEE для документации по тестированию программного обеспечения [829] дано хорошее описание тестовых документов и их взаимосвязи друг с другом и с процессом тестирования. К тестовым документам относятся, в частности, тест-план, спецификация тестового проекта, спецификация тестовой процедуры, спецификация тестового случая, журнал тестирования и отчет о тестовом инциденте или проблеме. Тестируемая программа с указанной версией и определенными требованиями к аппаратному и программному обеспечению, предъявляемыми до начала тестирования, документируется как элемент тестирования. Тестовая документация должна создаваться и постоянно обновляться по тем же стандартам, что и другая документация, находящаяся в разработке *(IEEE Computer Society, 2001, 92).*

Мы пытались разработать документацию в стиле IEEE Standard 829 и видели результаты попыток других компаний, работающих в различных отраслях. Мы не были довольны результатами. На самом деле, по нашему опыту, стандарт 829 принес больше вреда, чем пользы.

Частично наше разочарование в стандарте 829 связано с тем, что команды тестировщиков, пытаясь следовать ему, попадают в неприятные ситуации. Команда за командой создавала шаблоны планирования тестирования, основанные на стандарте 829, а затем бесполезные документы, соответствующие этому шаблону. Поначалу мы думали, что проблема в людях: они неправильно применяют стандарт. Позже мы пришли к выводу, что проблема скрыта более глубоко, поскольку широко распространена и с ней сталкивались люди, которых мы уважали.

Мы наблюдали следующий паттерн (или антипаттерн): команда тестирщиков создавала или заимствовала шаблон и вкладывала много сил в оформление документов, что приводило к получению первоначальной не очень информативной массы материалов. Затем участники команды сталкивались с затратами и ограничениями, связанными с таким типом документирования, и постепенно отказывались от него. В результате многим командам пришлось заниматься исключительно специальным тестированием, поскольку они потратили все свое запланированное время и бюджет на бумажную работу, результатами которой не воспользовались.

Отказ от усилий не означал публичного отречения от своей работы; обычно они молча переставали читать или обновлять материалы. Если

бы вы спросили их о документации тестирования, они, скорее всего, предъявили бы большую папку, полную документов (которые никто не читал и не обновлял). Несколько компаний проходили этот цикл раз за разом, ожидая, что в следующий раз они сделают работу лучше, и обвиняя себя в том, что у них это не получается.

Проблема *не в том*, что эти компании неправильно применяли стандарт 829.

Проблема *в том*, что он не мог удовлетворить требования этих компаний.

Чтобы эффективно применить решение, вам необходимо четко понимать проблему

Нигде этот принцип не проявляется так ярко, как при правильном и неправильном использовании документации тестирования. В связи с этим мы привели уроки, извлеченные обеими сторонами из дебатов по поводу соответствующих стандартов документации. Мы надеемся, что вы сначала подумаете о проблеме, которую должен решить ваш тестовый документ, а затем примените форму, соответствующую этому решению.

Не используйте шаблоны документации тестирования: шаблон не поможет, если он вам не нужен

Шаблон документации тестирования не заменит навыки.

Шаблон — это структура для создания документации тестирования. Заполните разделы или пустые поля — и у вас есть документация.

Проблема с шаблонами заключается в том, что они облегчают написание документа общего назначения, который внешне выглядит хорошо. Для некоторых людей это плюс. Но, чтобы использовать шаблон для написания хорошей документации тестирования, необходимо владеть им. Вы должны понимать, что означает каждый раздел, зачем он вообще нужен и когда его следует удалить. Если вы все это знаете, то шаблон вам не нужен. Если вы этого не знаете, то не используйте шаблоны. Они направят вас в контрпродуктивное русло, поскольку вы не поймете требования и компромиссы, которые пытался уравновесить автор шаблона.

Мы видели несколько организаций, которые создали шаблоны для себя, но использовали их неэффективно или контрпродуктивно. Человек,

использующий шаблон, должен уметь адаптировать его к конкретным требованиям.

Если вы можете написать эффективную документацию тестирования, не используя шаблон, то можете обнаружить, что шаблон помогает написать эффективную документацию быстрее.

Используйте шаблоны документации тестирования: они способствуют поддержанию постоянной коммуникации

Предположим, что вам необходимо представить свою работу третьей стороне. Возможно, вы составляете документацию тестирования, которая будет передана заказчику как часть продукта, разработанного по индивидуальному заказу. Дальше заказчик возьмет на себя сопровождение и тестирование. А может быть, вы составляете документацию тестирования, которую другая компания будет использовать для тестирования продукта. Возможно, вы составляете документацию тестирования, которая будет проверяться аудиторами или регулирующими органами или может быть использована в качестве судебного доказательства хорошего или плохого тестирования после того, как в результате аварии, связанной с продукцией вашей компании, погиб человек. В каждом из этих случаев передача материала будет более эффективной и людям будет легче им пользоваться, если вы применяете стандартный, предсказуемый формат, освещаете стандартный набор вопросов и используете стандартную терминологию.

Используйте стандарт IEEE 829 для документации тестирования

Проект стандарта IEEE 829 возглавлял Дэвид Гельперин. Это умный, вдумчивый, внимательный человек широких взглядов, который активно содействовал разнообразию, творчеству, развитию и применению навыков в области тестирования ПО. Его компания, *Software Quality Engineering* (SQE), организует конференцию STAR *(Software Testing Analysis & Review)* — одну из лучших и наиболее успешных конференций в этой области. SQE проводит и другие конференции. Мы регулярно выступаем на них. Компания также предлагает курсы для сообщества тестировщиков, и мы читали курсы для SQE. Наша критика стандарта 829 не является критикой Дэвида, которого мы считаем своим хорошим другом, или многих общественных услуг, которые SQE с гордостью предоставляет за свой счет сообществу тестировщиков ПО.

Многие атрибуты стандарта 829 отображают сильные стороны Дэвида. Ни одна из многочисленных категорий информации, содержащихся в стандарте, не является пустышкой. Читая стандарт, мы понимаем, почему кто-то хочет знать каждую часть информации, которую требует стандарт.

Ничто в стандарте не является обязательным. Если вы хотите создать тестовые сценарии в процедурном стиле, то стандарт 829 подскажет вам, как их назвать и где в документе их разместить. Если вы выбрали иной подход, то вам не нужно создавать документацию подобного типа только для того, чтобы соответствовать стандарту. Он предоставляет основу, структуру и набор определений, а не обязательный набор разделов.

Стандарт широко изучался и обсуждался. Люди, работающие в этой области, знают о нем. Он послужил основой для многих (возможно, большинства) шаблонов документации тестирования, которые циркулируют между компаниями. Он способствовал обсуждению планирования самого тестирования и его документации (так или иначе) во многих компаниях.

Двое из нас, Бах и Кейнер, выступали в качестве судебных консультантов (экспертов, выступающих в суде) в делах, связанных с некачественной продукцией, а наши коллеги выступали в других делах и давали нам подробные отчеты. Нам известны случаи, когда некачественная документация тестирования была частью основной проблемы, которая привела к судебным разбирательствам. Даже если документация не была ключевой проблемой, некоторые из компаний, якобы выпускавшие некачественную продукцию или обвинявшиеся в мошенничестве, могли бы выиграть, будь их документация тестирования более понятной и лучше структурированной. Беспорядок в ней ослабил их защиту. Стандарт IEEE 829 мог бы сэкономить этим компаниям много денег и, возможно, помог бы им сэкономить деньги своих клиентов.

Не используйте стандарт IEEE 829

Каждый из нас с энтузиазмом отнесся к стандарту IEEE 829, когда впервые прочитал его. Однако на практике мы столкнулись с рядом проблем. Когда мы комментировали эти проблемы, нам иногда говорили, что они связаны с неправильным использованием стандарта. Ведь он очень гибкий. Тестировщики не обязаны использовать стандарт не по назначению.

Для нас это похоже на аргумент типа «Не оружие убивает людей, а люди убивают людей». Иногда это весомый аргумент, иногда — нет.

Что касается случая с IEEE 829, то мы достаточно часто видели, как уважаемые нами люди сталкивались с проблемами в компаниях, которые работали с шаблонами на базе стандарта 829 в течение нескольких проектов. Поэтому мы считаем, что эти проблемы говорят о слабости стандарта и их не следует отвергать как отражение некомпетентности людей, использующих его.

Аргумент «Оружие не убивает людей» звучит по-другому, если речь идет о пистолете с чувствительным курком и без предохранителя, оставленном заряженным в общественном месте и снабженном официальной надписью: «Используйте этот пистолет для всех проектов».

Укажем ряд проблем, с которыми мы сталкиваемся, применяя стандарт на практике.

- Предположение, лежащее в основе стандарта, похоже, представляет собой каскадный подход, при котором тесты разрабатываются заранее, тщательно документируются, а затем не изменяются. Стоимость изменений (затраты на сопровождение документов) оказывает сдерживающее влияние на изменения. На наш взгляд, следует создавать новые тесты, которые будут усложняться, поскольку программа становится более стабильной и мощной по мере того, как вы получаете больше информации. В той мере, в какой стоимость сопровождения документации тестирования побуждает вас повторно использовать старые тесты вместо разработки новых, более мощных, и придерживаться текущей стратегии тестирования вместо того, чтобы совершенствовать ее по мере накопления опыта, документация тестирования является частью проблемы, а не частью процесса.
- Тестовые документы большого объема формируют менталитет соответствия. *Делай то, что написано в плане.* Такое мышление в корне отличается от менталитета бдительного, критически мыслящего тестировщика, который обращает внимание на любые подсказки и отслеживает все зацепки, представляющие интерес.
- Стандарт не содержит структуры для анализа требований к документации тестирования. В нем нет предложений или рекомендаций относительно того, когда и какую информацию следует представлять.
- Нет явного понимания и обсуждения (огромных) затрат на представление всех этих видов информации. Время, потраченное на эту документацию, вполне может оказаться временем, не потраченным на тестирование.

- В стандарте сделан упор на широту документации. Кажется, что больше — значит лучше. По-видимому, *неплохо было бы* подготовить тест-план, спецификацию проекта тестирования, спецификацию процедуры тестирования, спецификацию тестовых случаев и т. д.
- Не существует критериев, по которым можно было бы определить, хороши или плохи конкретные экземпляры документации тестирования. На практике объем, похоже, является общепринятой заменой ясности и покрытия. Нам приходилось рецензировать или проверять огромные тестовые документы, которые, по мнению авторов, были адекватными или полными, только для того, чтобы обнаружить зияющие дыры — ключевые стратегии тестирования или области риска вообще не были описаны. Слишком легко не заметить отсутствие даже самых очевидных тестов, когда документы занимают сотни или тысячи страниц.
- Затраты на сопровождение таких длинных документов огромны. При изменении программного обеспечения необходимо изменить не только те части документации, которые связаны с этим аспектом ПО. Приходится выполнять поиск во всех остальных частях, чтобы понять, что именно должно измениться. Эта работа выполняется параллельно поиску и изменению реальных тестовых файлов (файлов кода, если вы автоматизируете их). Если соответствие между документацией и кодом не 1 к 1, то будут возникать несоответствия, которые впоследствии приведут вас к неприятностям и временны́м затратам.
- Документирование каждого теста может серьезно помешать автоматизированному тестированию. Если на документирование одного тестового случая уходит час времени (мы считаем, что это заниженная оценка), а вы хотите, чтобы в вашем проекте было 10 000 автоматизированных тестов (это определенно *не завышенная* оценка количества отдельных тестов, используемых во многих компаниях), то на документирование придется потратить 10 000 часов работы тестировщика. Кроме того, какое-то время будет потрачено, чтобы проверить, точно ли автоматизированные тесты работают в соответствии с документацией. Когда тесты меняются из-за изменения тестируемого ПО, меняется и документация тестирования, что еще больше увеличивает трудозатраты тестировщика. Это чрезмерная плата за автоматизацию тестирования. Компании, которые следуют стандарту IEEE 829, будут писать меньше тестов и могут вовсе отказаться от них вместо того, чтобы нести расходы на их сопровождение и сопутствующую документацию. По нашему опыту, компании часто просто отказываются от документации, в результате чего

все усилия, затраченные до этого момента на документацию, пропадают втуне (поскольку она неполная и быстро устаревает).

- Парадигмы автоматизированного тестирования, предполагающие большие объемы тестов (миллионы), генерируемых или комбинируемых со случайными данными или случайными последовательностями, представляются совершенно чуждыми стандарту 829. Мы можем представить себе, что документация по программному обеспечению и связанные с ней модели ПО могут быть отнесены к категориям стандарта, но на практике этого не видим. Вместо этого мы наблюдаем, что эти работы (модели, код, оракулы и т. д.) документируются в другом месте или не документируются вовсе.
- Ранее мы уже рассказывали о нашем опыте судебных разбирательств. Завершим этот список проблем упоминанием другого опыта. Некоторые компании начинают с подписанного заявления о том, что процесс тестирования будет документироваться в определенной степени, будет соблюдаться шаблон (основанный на 829), стратегия тестирования будет такой-то и такой-то. Затем они завершают проект наполовину и отказываются от подписанного заявления в пользу, как им кажется на тот момент, реальной работы по реальному графику. Возможно, они приняли абсолютно правильное решение, но подумайте, как это будет выглядеть при судебном разбирательстве. Они начали с подготовленного плана и отраслевого стандарта, затем незаметно ослабили требования, а затем выпустили дефектный продукт. Это выглядит очень, очень, очень плохо. Если вы не собираетесь следовать слишком амбициозному плану, а ваша компания рискует подвергнуться судебному преследованию, то не начинайте свой проект с заявления о том, что вы собираетесь следовать этому плану. Чрезмерно амбициозный план может принести гораздо больше вреда, чем пользы.

Перечислив проблемы, вернемся к преимуществам. Понеся все эти затраты, затормозив наши проекты и побудив наших сотрудников мыслить более вяло и зашоренно, когда они проводят тестирование (а не пишут о нем), что мы получим взамен?

Многие компании используют значительно меньше бумаги. Они отслеживают состояние задач с помощью кратких списков и таблиц, отчетов о состоянии дел и регулярных командных совещаний. Они отслеживают свои проблемы с помощью хорошо написанных баг-репортов, хранящихся в хорошо управляемой системе отслеживания багов. Какие дополнительные преимущества получат эти компании, следуя стандарту 829? Насколько убедительны эти преимущества в *ваших* условиях?

Во многих ситуациях дополнительные преимущества не являются убедительными. В таких случаях, учитывая дополнительные затраты и риски, связанные с разработкой и поддержкой больших наборов документации тестирования, мы считаем, что создание документации в стиле стандарта 829 было бы безответственным.

Проанализируйте ваши требования, прежде чем принимать решение о том, какие продукты создавать; это относится как к документации, так и к программному обеспечению

Решение о том, что добавить в комплект документации тестирования, а что пропустить, нужно принимать, исходя из потребностей вашего проекта. Формат и категории стандарта IEEE 829 могут быть полезными, а могут и не быть. Пока вы не завершили анализ требований к документации, выбирать стандарт IEEE 829 (или любую другую подобную спецификацию) в качестве структуры и пользовательского интерфейса по меньшей мере преждевременно.

Мы не понимаем менталитет человека, который настаивает на том, что код никогда не следует писать, не проведя тщательный анализ требований, но при этом готов написать большой набор документации тестирования, обойдясь без соответствующего тщательного анализа требований.

Пожалуйста, не поймите нас неправильно. Мы не говорим, что стандарт IEEE 829 не подходит для вашего продукта. Он может быть идеальным. Точно так же, как COBOL может быть идеальным языком для вашего проекта. Мы лишь хотим сказать, что прежде, чем выбирать язык программирования и основные инструменты, подумайте о том, что собираетесь создавать.

Чтобы проанализировать требования к документации тестирования, задавайте вопросы

Мы рекомендуем книгу Гаусе и Вайнберга (Gause, Weinberg, 1989) в качестве подробного введения в анализ требований и как отличный сборник контекстно свободных вопросов, которые полезны для любого анализа требований. Михалко (Michalko, 1991, с. 138) предлагает дополнительный набор интересных контекстно свободных вопросов («во-

просы ЦРУ о Фениксе»). Помимо них, мы собрали набор вопросов, более характерных для требований к документации тестирования.

- **Какова цель вашей команды и каковы ваши задачи при тестировании данного продукта?** Документация тестирования (как и любой другой создаваемый вами результат работы) не имеет ценности, если не помогает вам достичь ваших целей.
- **Ваша документация тестирования — это продукт или инструмент?** Продукт — то, что вы даете другим, чтобы они могли этим пользоваться. Они платят за него. Вы, вероятно, будете следовать любому стандарту, соблюдения которого потребуют эти люди, при условии, что они готовы за это платить. В отличие от этого, если документация — лишь инструмент для внутреннего пользования, то она не должна быть более полной, структурированной или аккуратно отформатированной, чем необходимо для достижения поставленных целей.
- **Чем определяется качество программного обеспечения — юридическими вопросами или рыночными факторами?** Если ваше программное обеспечение и тестирование подлежат проверке со стороны регулирующих органов, то вы, вероятно, будете следовать официальному формату документации, например стандарту 829. Аналогично если ваш продукт может причинить вред людям или уничтожить имущество, то ваша документация тестирования может повлиять на судебный процесс. Формальная структура стандарта 829, традиционный объем документов в его стиле и статус «отраслевого стандарта» могут сделать стандарт 829 хорошим выбором. Солидная документация может помочь или не помочь повысить качество вашего программного обеспечения, но поможет вашей компании защитить себя впоследствии. С другой стороны, если следствием снижения доходности на вашем рынке является потеря продаж, а не судебные иски, то ваши клиенты никогда не увидят вашу документацию тестирования и не обратят на нее внимания.
- **Как быстро меняется дизайн?** Если дизайн программного обеспечения меняется быстро, то не стоит детализировать тесты: нюансы слишком быстро устареют. Не расписывайте процесс документирования тестов чересчур подробно: они будут пересматриваться или отменяться слишком быстро, чтобы оправдать вложенные в них средства.
- **Как быстро меняется спецификация в зависимости от изменений в проекте?** Нельзя проводить тестирование на основе спецификации, если она хронически неполная и устаревшая, и не

стоит привязывать документацию тестирования к содержимому такой спецификации. *(Примечание: остерегайтесь попыток делать именно это.)* Если проект меняется быстрее, чем спецификации, то, возможно, он нуждается в более совершенных спецификациях, а возможно, и нет. Неудобства для команды тестировщиков не самый весомый аргумент в пользу изменения политики спецификации в проекте. Если вы не получаете хороших спецификаций, то планируйте усилия по адаптации стратегии тестирования, а не политики проекта. Если вы собираетесь бороться за лучшие спецификации, то делайте это, исходя из затрат и рисков, создаваемых для других стейкхолдеров, особенно тех, кто играет более заметную роль в формировании прибылей и убытков компании.

- **Что вы надеетесь доказать во время тестирования — соответствие спецификации или несоответствие ожиданиям заказчика?** Если вы делаете заказное ПО по согласованной с заказчиком спецификации, то ваше тестирование и документация, скорее всего, будут ориентироваться на соответствие спецификации. Напротив, если ваш продукт предназначен для широкого потребления, то никто не подписывал спецификацию продукта, никакой контракт ее не поддерживает, и существует меньше гарантий того, что спецификация, которая есть у вас на руках, приведет к созданию продукта, удовлетворяющего техническим требованиям. В этом случае лучше использовать тесты с целью доказать, что продукт не понравится покупателям, чем для того, чтобы проверять его на соответствие имеющимся спецификациям. Подробная документация тестирования в этой связи будет содержать доказательства ожиданий заказчиков, например информацию о конкурирующих продуктах, общем оборудовании, которое будет использоваться с вашим программным обеспечением, критические обзоры данного продукта или его предшественников в прессе, а также другие материалы, ориентированные на заказчика и платформу.
- **Какой стиль тестирования вы предпочитаете — исследовательский или с использованием готовых тестов?** Если вы в основном используете уже готовые тесты, то вам может понадобиться документация по эксплуатации и сопровождению каждого из них. Если основной упор делается на исследовательское тестирование, то вам лучше предпочесть стратегическую и тактическую документацию (в которой содержатся идеи о том, какой подход выбрать для тестирования той или иной области, но не тестовые случаи), а также документацию по приобретенным или разработанным инструментам, облегчающим исследование.

- **Должно ли в документации тестирования указываться, что тестировать (цели) или как тестировать (процедуры)?** Мы предпочитаем документацию, в которой больший упор делается на цели, но пошаговые описания, безусловно, помогают представить процедуру тестирования третьим лицам.
- **Должен ли вообще проект тестирования осуществляться под управлением документации?** Хотите ли вы, чтобы тестировщики обращались к документации за оперативной информацией (например, за информацией о планировании, которая определяет, что делать дальше)?
- **Если проект осуществляется под управлением документации, то должно ли это происходить на всем его протяжении или только на отдельных этапах?** Должно ли тестирование проводиться в первую очередь со ссылкой на соответствующую документацию? Если да, то должно ли это правило соблюдаться на протяжении всего проекта или на ранних стадиях следует больше внимания уделять исследовательскому тестированию? А может быть, исследовательское тестирование должно проводиться на поздних этапах (для более тщательной проверки продукта, выглядящего рабочим)?
- **Кто является основными читателями этих тестовых документов и насколько важны эти люди?** Если вы хотите, чтобы тестировщики и программисты просматривали документацию (например, чтобы найти дыры в покрытии), то пишите ее так, чтобы акцентировать внимание на дизайне и облегчить понимание того, что охватывают тесты. Не делайте акцент на пошаговом описании тестов, иначе ваши читатели заблудятся.
- **Какая степень согласования вам необходима? К каким документам (спецификациям или требованиям) вы обращаетесь и кто их контролирует?**
- **В какой степени документация тестирования должна поддерживать отслеживание состояния проекта и хода тестирования и соответствующую отчетность?** Следует ли создавать систему обработки, позволяющую подсчитывать количество задокументированных и запланированных тестовых случаев, попыток и пройденных тестов и текущее количество найденных багов? Должна ли документация тестирования играть какую-то роль в этой системе? Например, должны ли тестировщики работать с документацией в интерактивном режиме в процессе тестирования, делая в ней пометки по мере выполнения задокументированных тестов? Следует ли собирать эти пометки или оценки состояния и сводить их в отчет о состоянии?

- **Насколько хорошо документация должна поддерживать делегирование работы новым тестировщикам?** Чтобы эффективно делегировать полномочия, необходимо объяснить человеку, что он должен делать, причем достаточно подробно, чтобы он смог выполнить эти действия. Эффективное делегирование не обязательно предполагает наличие пошаговых инструкций. Мы предпочитаем обучать новых тестировщиков некоторым навыкам, давать им вводные задания, которые знакомят их с документацией по продукту (например, с руководством пользователя), а затем давать им инструкции, предполагающие, что новички владеют этими навыками и справочными материалами. Вместо того чтобы замедлять работу (и увеличивать объем документации тестирования) для тестировщиков, которые не могут самостоятельно развить свои навыки и найти ответы, мы заменяем тех, кто не может справиться с данной задачей, теми, кто это может. Если вы считаете, что подробные инструкции необходимы, то мы предупреждаем, что написание эффективных подробных инструкций требует большого мастерства. Легкое введение в эти вопросы можно найти в книге Вурмана (Wurman, 1991). Еще один критический аспект делегирования — это ваша способность определить, что человек сделал и насколько хорошо. Если вы попросите кого-то сделать пометки в очень кратком документе (например, в одной из матриц, которые мы показывали в разделе «Дополнение к техникам тест-дизайна» главы 3), то сможете увидеть закономерности быстрее, чем если бы вам пришлось перелистывать десятки или сотни страниц сценариев тестирования, испещренных пометками.
- **Каковы ваши предположения о навыках и знаниях новых тестировщиков?** Все, что вы пишете, предназначено для аудитории. Чем больше ваша аудитория знает, тем меньше вам приходится ей рассказывать.
- **Используете ли вы документацию тестирования для документирования процесса проекта, для создания набора моделей или дескрипторов продукта, тестирование которых кто-то может проводить, или для того, чтобы дать читателю структуру, позволяющую обнаруживать баги?** Это совершенно разные цели для разных читателей с разными навыками и интересами.
- **Набор тестов должен обеспечивать предотвращение, обнаружение и предсказание. Что из этого наиболее важно для текущего проекта?** Если создать тест на ранних стадиях и вместе с программистами провести достаточно эффективное ревью, то они могут спроектировать программу с помощью подхода, который га-

рантирует, что она пройдет тест. Они думали о тесте во время разработки, поэтому не допустили возникновения бага. Просто проработав материалы и задав программистам ключевые вопросы, можно указать на риски и слабые места в их подходе. Это примеры профилактических преимуществ планирования тестирования. Позже вы получаете код. Если вы четко следуете плану, то он помогает вам найти имеющиеся баги. Таково преимущество обнаружения, свойственное планированию тестирования. Наконец, результаты тестирования могут помочь спланировать остальную часть текущего проекта или будущие проекты. Они могут выявить проблемные области, распространенные типы багов, а также эффективные и неэффективные стратегии. Они также позволяют получить статистику, с помощью которой можно определять время выполнения некоторых задач и, возможно, объема работ, оставшихся в данном проекте. *Ваши усилия по планированию тестирования принесут много пользы. Если бы вы могли сосредоточиться только на одном из этих трех направлений (предотвращение, обнаружение или предсказание), то каким бы оно было?* Разные команды имеют разные ответы на этот вопрос.

- **Насколько удобны в сопровождении тестовые документы (и их тестовые случаи)?** Насколько твердо они гарантируют, что изменения в тестах будут следовать за изменениями в коде? В одних компаниях спецификации продуктов являются документами-концепциями. Они помогают команде разработчиков создать первоначальный план, но никогда не обновляются. Документы, созданные позднее, помогают команде решать конкретные проблемы по мере необходимости. Другие компании создают спецификации, которые постоянно обновляются и тесно связаны с каждым аспектом продукта по мере его создания. *Какой из этих вариантов подходит именно вам?*
- **Помогут ли тестовые документы выявить (и пересмотреть или реструктурировать) постоянное изменение профиля риска программы?** Существует старая эвристика, согласно которой область программы с некоторым количеством багов содержит больше багов, чем обнаружено. Поэтому следует усиленно тестировать те области, где баги были обнаружены ранее. Однако в какой-то момент эта часть продукта может быть окончательно очищена. Другая часть продукта, которая раньше была работоспособной, может стать нестабильной. Собираетесь ли вы разрабатывать документацию тестирования таким образом, чтобы с течением времени можно было заметить изменения в стабильности различных областей продукта?

Задавая все эти вопросы, мы не призываем вас создавать документ требований и записывать в него все ответы. Мы предлагаем вам подумать над этими вопросами и записать ответы с такой степенью детализации, которая поможет вам выполнить задачи документации тестирования. Возможно, вам понадобится многостраничный отчет, в котором будут задокументированы ваши решения и который поможет вам утвердить их у руководства. В ряде компаний командам тестировщиков подобная документация нужна для защиты их работы, если проект сильно отстает от графика или если продукт демонстрирует неприемлемый уровень качества в условиях эксплуатации. В качестве альтернативы может быть достаточно сформулировать цель в одном предложении.

Обобщите свои основные требования к документации в одном предложении, состоящем не более чем из трех компонентов

Вот два совершенно разных резюме.

- Комплект документации тестирования поможет вам искать баги в этой версии, делегировать работу и отслеживать статус.
- Комплект документации тестирования позволит осуществлять непрерывное сопровождение продукции и тестирования в течение как минимум десяти лет, содержит учебные материалы для новых членов команды, а также позволит создать архивы, пригодные для использования в нормативных документах и при судебных разбирательствах.

Эти резюме приведут к созданию совершенно разных наборов документации. Просмотрите резюме с каждым заинтересованным участником проекта.

Глава 7

Взаимодействие с программистами

Бóльшая часть вашего общения с программистами будет связана с багами, о которых вы сообщаете. Этой важной области посвящена глава 4. В текущей же главе мы поговорим о других аспектах взаимодействия с программистами.

Программисты — эксперты в том, как думают машины, поэтому к ним часто относятся как к машинам. Не поддавайтесь этой модели поведения. Программисты не машины для кодирования. У этих людей есть чувства, и большинство из них очень заботятся о своей работе.

Многие программисты плохо относятся к тестировщикам. Мы считаем, что лучший способ предотвратить пренебрежение или противостоять ему — это построить личные отношения, основанные на взаимном уважении. Предположите, что люди, с которыми вы работаете, достойны уважения, и действуйте, исходя из этого. Выполняйте свою работу так, чтобы заслужить их уважение. Не допускайте неприемлемое отношение к себе и сами не ведите себя подобным образом.

Как официальный критик работы программиста, вы должны проявить чуткость и дипломатичность и выразить признательность. Не перегибайте палку и не подбадривайте, но дайте людям понять, что видите ценность их работы. Если их работа плоха, то не стоит говорить об этом в грубой форме.

Будьте открыты и честны в общении с программистами. Начните разговор. Возможно, вы захотите обсудить с ними эту главу. Прислушайтесь к тому, что они говорят.

Поймите образ мышления программистов

Все трое из нас начинали работать программистами, а затем специализировались на тестировании ПО. Мы до сих пор пишем код. Наш опыт влияет на то, как мы понимаем программистов и как, работая тестировщиками, мы работаем с программистами.

Программисты и тестировщики работают в разных условиях. Мы играем разные роли. Пребывая в них, мы по-разному относимся к своей работе. Вы можете быть более эффективными, если примете во внимание некоторые общие различия в наших взглядах и подходах.

Лучший способ научиться взаимодействовать с программистами — это стать одним из них и некоторое время поработать с другими программистами как со своими коллегами. Вернитесь к тестированию после того, как написали рабочий код, подвергшийся критике, осуждению и похвале со стороны тестировщиков, пользователей, руководителей и коллег-программистов. Все, что мы можем сказать в этой главе, меркнет по сравнению с пониманием, которого вы можете достичь благодаря такому опыту.

Обобщения, приведенные в этом уроке, применимы к разным людям по-разному. Мы настоятельно рекомендуем вам лучше узнать человека, с которым вы работаете, а не полагаться в первую очередь на эти наблюдения. Тем не менее некоторые из этих обобщений могут оказаться полезными.

- **Большинство программистов специализируются на узком круге задач.** Программист нередко сосредотачивается на подсистеме или модуле, полагаясь на часто отрывочную информацию о других элементах системы, с которыми должен взаимодействовать его код. Вы, напротив, часто являетесь специалистом общего профиля в отношении систем, которые тестируете. Чтобы тестировать хорошо, необходимо понимать, как все это сочетается. Конечно, вы можете предоставить программистам, с которыми работаете, информацию обо всей системе.
- **Программисты сосредоточены на своей теории системы.** У них есть модели того, как связаны компоненты системы, какие из них надежны и как распространяются ошибки. Программисты должны работать, исходя из своей ментальной модели. Когда они говорят вам, что баг, о котором вы сообщили, не может произойти, они не имеют

в виду, что считают себя непогрешимыми. Они говорят, что подобный баг не соответствует их модели и тому, что они считают истиной. Вы сосредотачиваетесь на наблюдениях и доказательствах. Это позволяет проверить модель программистов. Тщательно ведите записи и журналы, делайте в своих отчетах упор на том, что действительно видели, и позвольте программистам находить недостатки в их рассуждениях.

- **Программирование — сложная деятельность.** Программисты тратят много энергии на то, чтобы понять создаваемые ими системы. Такая концентрация часто мешает им уделять внимание вещам, которые вы считаете важными, и вдобавок понуждает их нервно реагировать в ситуациях, когда их отвлекают от работы.
- **Программисты часто оказываются в сложных ситуациях.** Они имеют дело с неоднозначными и меняющимися требованиями, несовершенными инструментами и технологиями компонентов, а также со средой, работа которой часто прерывается.
- **Многие программисты не любят рутинную работу и часто создают инструменты и скрипты для автоматизации повторяющихся задач, с которыми им приходится сталкиваться.** Многие рассматривают тестирование как повторяющуюся задачу, которую, естественно, следует автоматизировать. Они могут предположить, что с вами что-то не так, если вы не автоматизируете свои тесты. Не покупайтесь на это. Не пытайтесь автоматизировать тесты только для того, чтобы завоевать уважение программистов. Есть более эффективные способы. Ваша честность и компетентность потребуют уважения (см. урок 153).

Более подробно об этом см. статью Петтикорда (Pettichord, 2000b) под названием *Testers and Developers Think Differently*.

Развивайте доверие программистов

Не создавайте излишне враждебных отношений с теми, кто создает программы, которые вы тестируете. Вы будете работать более эффективно, если программисты, с которыми вы взаимодействуете, станут делиться с вами информацией, например своими планами, черновиками проектной документации и прототипами. Выясните, какого рода обратная связь им нужна, и дайте ее им.

Чем раньше вы сможете привлечь программистов, тем лучше для вас. Раннее взаимодействие требует от вас чуткости и готовности помочь. Когда вы имеете дело с черновым вариантом кода, программисты знают

о существовании проблем. Они не хотят слушать о том, что им известно, например, о необходимости обработки ошибки, которую они еще не написали. Они хотят знать о серьезных проблемах. Выясните, что они считают серьезным, и сосредоточьтесь на информации, которую вы им предоставляете на данный момент. По мере развития проекта у вас будет время разработать более независимые критерии или повлиять на них.

Если вы не согласны с мнением, что некий элемент приведет к проблемам, то выскажите свое мнение, но не придирайтесь. Правы вы или нет, станет понятно позже.

Предоставляйте услуги

Предлагайте помощь непосредственно программистам. Это укрепляет доверие и доказывает, что с вами стоит сотрудничать. Ниже указаны некоторые услуги, которые вы можете предложить.

- Тестирование компонентов сторонних производителей. Делитесь результатами тестирования, чтобы программисты могли решить, можно ли использовать эти компоненты в продукте и каким образом.
- Тестирование частных сборок и прототипов.
- Настройка тестовых сред для программистов, которые они могут использовать, проводя собственное тестирование.
- Анализ документов с требованиями на предмет проверяемости. Программисты с трудом справляются с неоднозначными требованиями. Они могут быть очень рады вашему участию.

Все, что вы делаете как тестировщик, должно быть предоставлением услуг. Выше мы привели примеры лишь прямых и очевидных услуг. Они дают вам возможность завоевать доверие программистов, а также продемонстрировать свою компетентность.

Ваша честность и компетентность потребуют уважения

Вы защищаете интересы клиентов. В конечном итоге ваша работа заключается в том, чтобы сообщать о проблемах, с которыми сталкиваются пользователи. Программисты и руководители могут не хотеть признавать эти проблемы. В этом случае вы отправляете неприятное сообщение. За это вас могут не любить. Но если вы находите проблемы,

которые заслуживают внимания, и сообщаете о них точно и прямо, вас будут уважать. Сообщая о проблемах, следуйте этим рекомендациям.

- **Четко излагайте проблемы.** То есть описывайте баг пошагово, не указывая лишние этапы (или приводя небольшое их количество). Описывайте симптомы отказа точно. Сделайте свой отчет простым и понятным. Ваша работа будет оценена по достоинству, поскольку вы проявляете уважение к времени программиста (как читателя баг-репорта и как исследователя, который полагается на ваш отчет в поисках подсказок).
- **Основывайте свои суждения на реально наблюдаемом поведении продукта.** Зачастую вы пользовались программой больше, чем кто-либо другой. Это делает вас экспертом по внешнему поведению программы. Вы не являетесь экспертом по внутреннему устройству. Говорите о том, что видите, и не тратьте много времени на свои предположения о природе проблемы.
- **Если отказ не поддается воспроизведению, покажите, какую работу вы проделали, пытаясь его воспроизвести.** Когда вы отправляете отчет о невоспроизводимом баге, лучшее впечатление, которое может создаться, — что вы провели тщательное расследование, однако необходимы более совершенные инструменты и информация, чем те, которыми вы располагаете. Худшее впечатление, которое может возникнуть, — что вы при первой же трудности сдались и свалили работу на программиста. Проявляйте уважение к его времени.
- **Сообщайте плохие новости напрямую.** Не обращайтесь к начальству программистов, пока не дадите им возможность принять меры. Скажите им, что собираетесь обострить проблему, прежде чем сделать это.
- **Не притворяйтесь, что знаете то, чего не знаете.** Например, если вы не знаете, насколько серьезна проблема, то не притворяйтесь. Либо получите доказательства (например, от службы технической поддержки или сотрудников отдела маркетинга), либо промолчите, либо четко дайте понять, что высказываете лишь предположение.
- **Не усложняйте свои баг-репорты.** Но и не упрощайте их, а также не поддавайтесь на запугивания и уговоры игнорировать или скрывать то, что видите. Стойте на своем: увидев проблему, сообщите о ней; если сочтете нужным, то переведите ее в более серьезную плоскость. Создайте себе репутацию прямолинейного человека — и завоюете уважение.
- **Если вас считают честным, то вы можете развивать свою компетентность.** В противном случае она не будет иметь значения.

Сосредоточьтесь на работе, а не на человеке

Если видите баги, то сообщайте о них. Не сообщайте, что программист Джо работает из рук вон плохо. Может быть, так и есть, но если вы скажете об этом, то тем самым навредите *своей* эффективности.

Многие опытные тестировщики используют свои наблюдения за недостатками работников и организации в целом, чтобы сосредоточиться на поиске багов. Они видят беспорядок и предсказывают, что он может вызвать появление бага. Вы можете быть вполне успешны в такой деятельности, и это натолкнет вас на мысль, что будет проще и эффективнее сообщить о проблемах, которые вы видите в организации, напрямую руководству. *Ошибаетесь!*

Как только вы возьмете на себя задачу сообщать о проблемных программистах, все они перестанут делиться с вами информацией и приглашать на свои встречи. Это сделает вас неэффективными и превратит в часть проблемы.

Не стоит недооценивать способность руководства замечать эти трудности. Проблемы с людьми гораздо легче заметить, чем исправить. Помните, что вы не являетесь руководителем человека, который (на первый взгляд, а может быть, и на самом деле) игнорирует проблемного сотрудника. Привлекая внимание к возможной некомпетентности программиста, вы ограничиваете возможности руководства в том, как с ней справиться. Или же вынуждаете руководителей столкнуться с проблемой, которую они старались не замечать. В любом случае вы будете в проигрыше.

Некоторые тестировщики доходят до того, что считают своей обязанностью наказывать программистов за ошибки, пропуски сроков и несоблюдение технологического процесса. Угадайте, что происходит с такими тестировщиками? Они становятся лишними. Одних выдворяют сразу. Других держат в качестве удобных «плохих полицейских» до тех пор, пока вследствие какой-нибудь очень большой ошибки не потребуется принести в жертву козла отпущения.

Если вы видите закономерность в возникновении проблем, которые, как вы опасаетесь, не решаются, то незаметно представьте доказательства соответствующему руководителю, и пусть он займется этим вопросом. Вы выполнили свою работу (Pettichord, 2001c).

Программисты любят рассказывать о своей работе. Задавайте им вопросы

Многие тестировщики отмечают, что им трудно получить информацию от программистов. Мы обнаружили, что программисты часто охотно рассказывают о своей работе.

Лучше всего начать с проектной документации, с которой они работают. Начните с наведения справок. Прочитайте имеющиеся документы. Если есть возможность, то посмотрите код.

Иногда содержимое документов может быть запутанным. Спрашивайте программистов о тех разделах, которые вам кажутся важными, но непонятны. Иногда задавать вопросы можно с помощью электронной почты, но часто приватная беседа оказывается более эффективной, особенно если вопросы требуют уточнения. Если программисты согласятся на встречу, то придите подготовленными, чтобы не тратить их время.

Если у них нет документов, то попросите фотографию системы. Большинство программистов держат в голове образ системы, над которой работают, и будут рады поделиться им.

Можно попросить их составить схему системы на доске. Один из приемов заключается в том, чтобы указать на случайную стрелку или квадрат и спросить: «Что произойдет, если это не сработает?» Это может выявить недостатки в обработке ошибок или предположения, не вызывающие сомнения. Подобные вопросы, заданные двум или более программистам, могут выявить интересные различия в их взглядах.

Зачем вы задаете эти вопросы? Чтобы больше узнать о создаваемой системе, о том, как она может выйти из строя, а также об отношении к ней и предположениях тех, кто ее создает. Не проверяйте и не допрашивайте их. Если они почувствуют, что это происходит, они не будут сотрудничать.

Получив ответы, сделайте свои заметки и поделитесь ими с программистом и другими тестировщиками. Программистам не нравится снова и снова отвечать на одни и те же вопросы разным тестировщикам.

Очень полезно знать язык программистов. Если они работают на C++ или Java, то вы должны иметь представление о том, что такое класс.

Если ПО работает в многопоточной системе, то вы должны знать, что такое поток.

Активное слушание само по себе может принести много пользы. В обычном разговоре каждый человек высказывает свои идеи, делится опытом или соображениями. Активно слушая, вы стараетесь помочь собеседнику выразить то, что он хочет сказать. Вы можете пересказывать слова собеседника, используя свои термины, задавать вопросы, позволяющие получить дополнительную информацию и делать выводы.

Ваша работа как тестировщика заключается в том, чтобы думать о том, как продукт может выйти из строя. Но как член команды вы должны понимать, в чем ценность создаваемого продукта. Дайте программистам понять, что вы понимаете ценность их действий.

Не говорите программистам, что они должны представить определенные документы, прежде чем вы сможете выполнить свою работу. Если они представляют проекты документов, в которых опущены важные детали, то спрашивайте. Если вам нужна информация, то скажите об этом. Объясните, зачем она вам нужна и как поможет вам в работе. Программисты не могут читать ваши мысли. (См. Gause, Weinberg, 1989, глава 6; Michalko, 1991, глава 14.)

Программисты рады помочь улучшить тестируемость

Большинство программистов хотят, чтобы их программы хорошо тестировались. Они стараются делать свою работу качественно, знают, что могут допускать ошибки, и ожидают, что вы найдете их.

Для тестировщика в понятие «тестируемость» входит все, что облегчает тестирование ПО. При общении с программистами более практичным определением является следующее: *тестируемость подразумевает наблюдение и контроль* (см. урок 137). Это определение указывает на природу возможностей, которые помогут вам. Понимая это, программисты могут предложить функции, о которых вы не думали (или не просили), но которые будут полезны. О чем следует просить? В уроке 137 приводится ряд примеров.

Многие тестировщики разочаровывались в своих попытках добиться от программистов информации о возможностях тестирования. Мы считаем, что есть три ключевых момента, которые помогут добиться успеха при озвучивании таких просьб.

- **Говорите на их языке.** Уметь читать проектную документацию и код полезно. Вы должны излагать свои просьбы в понятных программистам терминах. Если вы точно укажете, какой интерфейс вам нужен в той или иной части кода, то они честно выслушают вашу просьбу. Более того, вы можете с удивлением обнаружить, что они уже добавили возможности, подобные тем, о которых вы просили, чтобы помочь с отладкой или по другим причинам.
- **Спрашивайте заранее.** (См. урок 138.)
- **Будьте реалистами.** Одни запросы на тестируемость будут достаточно малы, чтобы их можно было запланировать вместе с другими задачами реализации. Другие представляют собой новые возможности, и их нужно заложить в бюджет и запланировать. Вам также придется отстаивать их необходимость, выступая перед руководством.

Большинство программистов любят программировать. Обращаясь с конкретной и разумной просьбой, вы даете им шанс впечатлить вас. Что им не нравится, так это когда им приходится читать мысли людей. Они не примут еще одну двусмысленную просьбу.

Многие тестировщики рассказывали нам, что программисты отвечают отговорками: «Это нарушит безопасность программного обеспечения», «Это снизит производительность». Иногда это вполне обоснованные опасения по поводу тестового кода. Но обычно, как мы полагаем, это кодовые фразы, означающие «мы не хотим об этом думать». Для того чтобы убедить программистов в том, что помощь вам в конечном счете выгодна им самим, может потребоваться применить определенные навыки продаж. Возможно, вам придется найти правильного программиста, обладающего ви́дением и влиянием, который сможет сделать это. Тем не менее если знать, о чем просить, делать это в нужное время и вежливо, то, на наш взгляд, можно добиться конструктивного рассмотрения своих запросов.

Мы много раз видели, как расширение возможностей тестирования способствовало улучшению продукта. Что-то было сделано по просьбе одного из нас. Что-то предложено тестировщиками или программистами команд. Это может быть трудным делом, но оно того стоит. Мы поощряем настойчивость.

Глава 8

Управление проектом тестирования

Управление проектом тестирования в некоторых отношениях похоже на управление любым другим проектом. Но есть по крайней мере одна особенность проекта тестирования: он определяется проектом программирования. То, что делаете вы, является реакцией на то, что делают программисты. Именно поэтому использование такого инструмента, как Microsoft Project, для планирования задач тестирования может сильно разочаровать. Вам придется изрядно постараться, чтобы уместить работу вашей команды в эти маленькие столбцы диаграммы Ганта. В этой главе мы рассмотрим уроки, которые мы извлекли из динамики проектов тестирования, и то, как ее контролировать.

Создайте культуру обслуживания

Проектные команды разрабатывают программное обеспечение в целях предоставления заказчикам определенных преимуществ. Заказчики могут быть внутренними или внешними, платящими или неплатящими. Заказчиками могут быть те же люди, что и разработчики (например, когда мы создаем собственные инструменты).

Тестировщики оказывают услуги в рамках всего проекта. Типичная услуга — поиск багов и составление баг-репортов. Другие услуги зависят от целей вашей команды (см. главу 1).

Один из фундаментальных вопросов, проходящих через всю литературу по тестированию и субкультуры тестировщиков, — какова ваша роль: обслуживающая или контролирующая.

- Поставщик услуг контролирует качество и соответствие предоставляемых им услуг общим усилиям по выработке конечного результата. *Мы предоставляем отличные услуги людям, которые в них нуждаются.*
- Поставщик услуг не контролирует качество конечного продукта, не контролирует процессы, используемые другими поставщиками услуг (программистами, писателями, маркетологами), не утверждает выпуск продукта и не отказывает в утверждении. Поставщик услуг не является руководителем проекта, он оказывает ему услуги.

Не пытайтесь создать культуру контроля

Тестировщики часто получают и дают подробные советы о том, как следует управлять проектами. Мы считаем, что многие из этих советов наивны. Худшие и наиболее догматичные из них исходят от тестировщиков и консультантов, чей опыт разработки продуктов ограничивается тестированием (либо иным измерением или оценкой) работы других людей.

Тестировщики часто видят проекты в их худшем виде. Эти люди сталкиваются с последствиями плохих и неполных решений и наполовину выполненных задач, не замечая планирование и работу, которые позволили продукту стать настолько хорошим, насколько он им является. То, что кажется плохими и неполными решениями, часто является продуманными бизнес-решениями, с которыми вы не согласны. Легко поверить в то, что вы знаете больше и сделали бы это лучше.

Некоторые процессы облегчают или затрудняют работу тестировщиков. Однако компетентная команда тестировщиков может оказать важные услуги широкому кругу руководителей, которые используют совершенно разные стили управления, в том числе создающие затруднения для тестировщиков.

Некоторые процессы, похоже, предназначены для выпуска некачественной продукции. Это серьезная проблема, и кто-то должен ею заниматься. К сожалению, *хуже всего* с этим справляется команда тестирования. У таких команд нет ни ресурсов, ни опыта, ни политической власти, чтобы исправить более широкие процессы разработки или управлять фиксированными процессами.

Мы не говорим, что вы (человек, который сегодня проводит тестирование) должны знать свое место и не лезть не в свое дело. Это далеко не так. Мы

призываем вас расширять свою роль и влияние в компании. Если вы хотите управлять деятельностью руководителя проекта и обладаете нужными компетенциями, то делайте это. Но делайте это из соответствующей роли — руководителя проекта. Это не роль менеджера по тестированию.

Укрепляйте свое влияние

Ваше влияние в компании зиждется на ваших исследовательских навыках и свободе общения. Оно не зависит от порядка субординации, поскольку вы не занимаете высокого положения во властной вертикали проекта.

В идеальном случае вы можете указать на проблему любой команде в компании, которую она затрагивает. Серьезная проблема, которая остается нерешенной, будет стоить некой команде (командам) больших денег. Если участники команды считают, что проблема достаточно серьезна, то будут добиваться ее решения. Если не считают ее таковой, то, возможно, ее не удастся решить. Не все проблемы будут решены.

В одних компаниях (по нашему опыту, в нескольких) считается само собой разумеющимся, что тестировщики будут демонстрировать свои баги и отчеты об их статусе всем, кто их будет слушать. Руководители проекта раздражаются, жалуются, а их коллеги советуют им смириться: тестировщики останутся теми, кто они есть.

В других компаниях иерархия соблюдается более строго. В их культуре не принято, чтобы тестировщик приходил к маркетологу или менеджеру по технической поддержке и обсуждал проблемы с разрабатываемым продуктом.

Если ваша задача — помочь компании принять правильные бизнес-решения в отношении обнаруженных дефектов, то ваша эффективность будет ограничена, если вы не сможете сообщить о своих выводах тем сотрудникам компании, на которых эти дефекты оказывают наибольшее влияние. Если у вас нет возможности донести информацию напрямую, на встречах один на один, то можете сообщить о них косвенно. Например, можно предоставить людям права доступа к базе данных отслеживания багов и научить их использовать ее для поиска проблем, которые могут их заинтересовать. Или можно публиковать отчеты о состоянии дел, в которых освещаются ключевые проблемы. Когда люди из других команд начнут ценить предоставляемую вами информацию и полагаться на нее, они (зачастую) будут готовы отстаивать ваше право предоставлять ее им.

Вы должны оценивать культуру своей компании и работать в тех рамках, которые не приведут к увольнению или остракизму. В этих рамках мы предлагаем вам использовать и развивать свое влияние, являясь авторитетным, высокоинтеллектуальным источником информации, которую ценят люди. По нашему опыту, таким образом можно добиться гораздо большего фактического влияния, чем с помощью процедурных полномочий, таких как право отказать в подписании (утверждении) выпуска.

Вы руководите подпроектом, который предоставляет услуги по тестированию, а не проектом разработки

Работа по тестированию — подпроект общего проекта. Вы используете ресурсы и предоставляете услуги. Вы в значительной степени контролируете ход выполнения проекта по тестированию и должны выбирать свой стиль так же тщательно, как, возможно, хотели бы, чтобы руководитель проекта выбирал свой.

Иногда руководители проекта совершают ошибки, а иногда им может пригодиться ваш совет. Во всяком случае, предлагайте их. Высказывайте свое мнение. Но окончательное решение о том, как вести проект, принимает его руководитель. Если он не прислушивается к вашим советам, значит, так тому и быть.

В этой главе мы не будем много писать о человеческих проблемах, связанных с управлением людьми в стрессовых ситуациях. Более подробно мы рассмотрим эту тему в главе 9. Пока же отметим, что, принимая неверные решения, руководители проекта и руководство иногда оскорбляют тестировщиков, требуют работать сверхурочно и порочат их честность. Такие действия выходят за рамки полномочий руководителя проекта. Важной частью вашей работы как менеджера по тестированию является защита сотрудников от унизительного отношения к ним.

Все проекты развиваются. Хорошо управляемые — развиваются активно

На протяжении каждого проекта следует ожидать (как нормального явления) больших и малых уточнений или корректировок общего плана.

Проект — это совокупность задач. Со временем команда обнаружит, что одни задачи сложнее или требуют больше времени, чем ожидалось;

другие пока не могут быть выполнены, поскольку ключевой специалист, отвечающий за эту задачу, занят чем-то другим; третьи по-прежнему кажутся менеджеру по маркетингу или заказчику более или менее срочными, чем на прошлой неделе. Кроме того, каждый раз, когда вы отправляете баг-репорт, вы добавляете новую задачу.

Проект обеспечивает структуру, позволяющую интегрировать новую информацию и определить, какие (вероятные) задачи следует решать дальше и в последующих итерациях, пока продукт не будет завершен. Рассматривайте проект как постоянный разговор о том, что имеет смысл делать дальше.

Поздние изменения будут всегда

Многие традиционные подходы к управлению проектами направлены на ограничение и контроль изменений, но другие принимают их (например, см. Weinberg, 1992; Beck, 1999; Beck et al., 2001; Krutchen, 2000). Тем не менее все подходы к управлению проектами должны учитывать изменения.

Представьте, что делаете новый стул взамен износившегося. Вам понятно, что для этого нужно, кому нужен данный предмет мебели, что с ним будут делать эти люди и какие нагрузки будут оказываться на стул. Вы можете найти людей, которые сделали стулья, очень похожие на тот, который вы собираетесь изготовить.

С программным обеспечением все не так. В большинстве программных проектов никто ранее не создавал именно такой продукт, а если и создавал, то люди, работающие над этим проектом, в этом не участвовали. Кроме того, люди, которые будут использовать это ПО, раньше не имели с ним дела. Даже если у них есть представление о том, что им нужно, они не знают, как сформулировать свои требования, в силу следующих причин.

- Заказчики не знают всех своих требований.
- Требования будут меняться по мере того, как заказчики будут пробовать ранние версии ПО или продукты конкурентов. Они будут открывать для себя новые способы использования ПО и представлять себе другие варианты его применения, которые хотели бы реализовать, но пока не могут.
- Разные стейкхолдеры имеют различные потребности, которые часто противоречат друг другу. Ни один документ не может сформулиро-

вать все противоречивые и потенциально противоречивые требования и сбалансировать их.

Более того, по мере создания компонентов и инструментов, а также приобретения навыков ожидаемые затраты на предоставление определенной выгоды будут меняться, делая более или менее легким удовлетворение потребностей тех, кто хочет получить эту выгоду.

Требования — это результат постоянной борьбы между тем, что мы хотим, и тем, что мы можем иметь (Bach, 1999a). По мере реализации проекта требования меняются.

Проекты предполагают компромисс между функциями, надежностью, временем и деньгами

Задача руководителя проекта — предоставить нужный набор функций с соответствующим уровнем надежности, в срок и в рамках бюджета. Это сложный компромисс.

- **Функции.** Выберите правильный их набор, учитывая, что предоставление всего, что может потребоваться каждому стейкхолдеру, обойдется слишком дорого.
- **Надежность.** Сделайте так, чтобы продукт работал, но остерегайтесь тратить бесконечное количество времени и денег, стараясь сделать так, чтобы он работал идеально при всех возможных обстоятельствах.
- **Время.** Как можно скорее запустите продукт в промышленную эксплуатацию или выставьте на продажу.
- **Стоимость.** Создайте продукт, сведя к минимуму уровень разумных издержек. В стоимость входят денежные и альтернативные расходы. Если в своем проекте вы используете критический ресурс (специалиста или уникальную машину), то он недоступен для другого проекта.

Мы характеризуем все это как компромисс, поскольку руководитель проекта может получить больше по одному критерию, затратив больше ресурсов в другом или вовсе пожертвовав им. Например, набор функций будет больше, если вы потратите больше времени (и/или денег) на создание продукта.

Если вы хотите понять странное решение руководителя проекта, то попробуйте разобраться, как оно влияет на каждый из четырех критериев.

Позвольте руководителю проекта выбрать его жизненный цикл

Модель жизненного цикла — это описание процесса проектирования и разработки продукта с того момента, когда кто-то впервые задумался о его создании, до выпуска в свет и ввода в эксплуатацию.

Мы не признаем удачной идею единственного оптимального варианта жизненного цикла разработки программного обеспечения. В каждой модели упор делается на одних аспектах проекта, а другие остаются открытыми для доработки. Модели жизненного цикла различаются тем, что именно остается открытым.

Некоторые компании придерживаются стандартной модели жизненного цикла, но, по нашему опыту, у руководителя проекта всегда есть возможность вносить изменения. Мудрый руководитель выбирает подход, позволяющий контролировать моменты, которыми ему трудно управлять в текущем режиме, и оставляет открытыми вопросы, в которых он особенно силен. Любой выбор чреват рисками и последствиями. Выбор не всегда удобен для команды тестирования, и не всегда это тот выбор, который сделали бы вы. В той мере, в какой он есть, выбор остается за руководителем проекта.

В следующих двух уроках мы рассмотрим водопадный и эволюционный жизненные циклы. Водопадный очень часто пропагандируют консультанты по тестированию, однако он не всегда отвечает нашим интересам. Эволюционная модель часто лучше подходит для тестировщиков, но может усложнять работу других членов команды проекта.

Мы не утверждаем, что вы должны выбирать исключительно эволюционную модель вместо водопадной. Мы не предлагаем настаивать на использовании какого-либо одного жизненного цикла. Мы говорим лишь о том, что распространенные варианты жизненных циклов сильно отличаются друг от друга и что выбрать один из них не так просто.

В водопадных жизненных циклах надежность противопоставляется времени

Водопадная модель описывает определенный жизненный цикл управления проектом, который проходит по таким фазам, как:

- определение проблемы (что мы пытаемся создать и почему);
- определение требований;

- создание внутреннего и внешнего дизайна;
- написание кода;
- тестирование;
- установка;
- поддержка после установки;
- судебные иски от разочарованных клиентов;
- определение проблемы (для следующей версии продукта).

Свое название модель получила благодаря фазовой диаграмме, которая для некоторых людей выглядит как водопад. Термин *«водопад»* описывает линейный характер этого подхода. Вы переходите от одной фазы к другой, и вернуться к предыдущей фазе очень сложно, так же как вода не может двигаться вверх по водопаду.

На практике определенное количество возвратов происходит (мы обнаруживаем ошибки или невозможности в заявленных требованиях, когда пытаемся спроектировать продукт, который будет им удовлетворять), но при контроле изменений такие действия пресекаются.

Водопадный подход имеет разновидность — V-модель. В ней описывается более широкая роль тестирования. В конце каждой фазы тестировщики формально оценивают результат работы. Например, в конце фазы определения требований тестировщики проверяют (и, предположительно, утверждают) документы по требованиям. Они также пишут ряд тестов, которые должны быть выполнены после того, как последующие фазы будут завершены и соответствующие части системы начнут работать[1].

---

[1] У нас есть еще несколько опасений, связанных с V-моделью. Процесс написания подробных тестов перед доставкой кода является рискованным. По мере изменения дизайна продукта тесты устаревают. К моменту завершения кода многие из написанных вами тестов будут описывать те аспекты продукта, которые никогда не были написаны или были написаны совершенно иначе, чем предполагалось изначально. Во многих проектах вся эта бумажная работа (даже хуже, если есть тестовый код и обширные тестовые данные) ничего не дала. Они устарели еще до того, как принесли какую-либо пользу. Сторонники V-подобных процессов говорили нам, что эти результаты работы команды тестировщиков могут быть использованы программистами, чтобы помочь им обнаружить двусмысленности и слабые места в предлагаемой функции до ее написания. Мы, конечно, согласны с тем, что анализ проектов может улучшить их и предотвратить проблемы. Но если цель состоит именно в этом, то мы полагаем, что проверка дизайна и ревью кода могут внести гораздо больший вклад в качество продукта, причем за гораздо меньшее время, чем предварительное написание тестов, которые никогда не будут выполняться. И вместо того, чтобы ждать окончания фазы, мы предлагаем проверять дизайн и код по мере того, как они становятся доступными для ревью.

Водопад (с буквой V или без нее) выглядит как аккуратный процесс, но что происходит с проектами, в которых не все действия выполняются на той или иной фазе? Что делать, если проект значительно отстает от графика?

Большинство проектов по разработке ПО действительно сильно отстают от первоначального графика. Мы встречали советы опытных специалистов, сводящиеся к тому, что «в хорошо управляемых проектах такой проблемы не будет». Однако проекты по разработке ПО несут в себе значительный риск. *Поздние изменения будут всегда.* Говорить: «Этого не случится» — значит выдавать желаемое за действительное.

Что *происходит*, если проект значительно отстает от графика?

При водопадном подходе к тому времени, когда вы получаете код, все функции уже спроектированы и большинство или все из них написаны. Бóльшая часть средств на разработку программного обеспечения уже потрачена. Ключевой компромисс — между временем и надежностью: исправить баги и выпустить продукт с опозданием или выпустить его раньше, зная, что в нем много багов.

Этот компромисс — традиционный объект жарких споров руководителей проекта и тестировщиков. Многие тестировщики требуют более строгого следования водопадной модели управления проектами. Это не решение проблемы. Компромисс между надежностью и временем на поздних стадиях проекта является неотъемлемой частью водопада. Укрепляя стенки коробки, вы не выберетесь из нее.

Хорошо подумайте, прежде чем ратовать за водопадный цикл или его вариант (V-модель).

В эволюционных жизненных циклах функции противопоставляются времени

При эволюционном подходе к разработке программного обеспечения команда проекта добавляет по одной функции за раз. Они проектируют функцию, пишут ее, тестируют и исправляют. Когда продукт с интегрированной в него функцией удовлетворяет стандартам качества команды, добавляется следующая функция (Gilb, 1997; Beck, 1999).

Команда разработчиков может выпустить этот продукт (последнюю версию, прошедшую тестирование) в любое время. Разница между

сегодняшней версией и версией, которая выйдет в следующем месяце, заключается в том, что в следующей версии будет больше функций. Обе версии работают. В конце проекта не существует компромисса между временем и надежностью.

У такого подхода есть свои проблемы. Представьте, что вы менеджер по маркетингу или технический писатель. Подойдите к руководителю проекта и спросите: «Какие функции есть в этом продукте?» Ответ, конечно же, будет таким: «Это зависит от того, когда мы его выпустим». Тот, кто должен знать, что будет в продукте на момент его выпуска, найдет эволюционный подход сложным. Некоторые считают, что с проблемой неопределенного набора функций и рисками ползучего расширения функциональности легче справиться в рамках водопадной модели, чем используя эволюционный подход.

Будьте готовы выделять ресурсы на проект в ходе ранних этапов разработки

Становится общепринятым мнение, что тестировщиков следует привлекать к работе на более ранних этапах цикла разработки. Однако команды тестирования, как правило, недоукомплектованы и перегружены работой. Чем же так привлекательны результаты, которые можно получить на ранних этапах разработки, что ради них стоит отрывать тестировщиков от текущих кризисных дел?

- Если вы просто отправляете тестировщика на совещания, проводимые на ранних этапах проекта, то, скорее всего, вы зря тратите его время.
- Если тестировщик не владеет языком программирования, то посылать его на ревью кода — пустая трата времени, сопряженная с риском для него продемонстрировать свое невежество и потерять уважение программистов.

Есть мероприятия, в которых тестировщики *могут быть полезны* на ранних этапах разработки. Приведем несколько примеров.

- Проверка любых документов с требованиями на предмет их понятности, проверяемости и однозначности.
- Проведение тестирования других артефактов проекта (документов, кода и т. д.) по мере их разработки. Не ждите, пока все будет

готово. Начинайте работать с артефактом, когда, по словам автора (которым вы верите), он достаточно проработан для того, чтобы его можно было подвергнуть полезному тестированию или ревью.

- Облегчение ревью кода. Ревью — серьезный шаг по улучшению качества. Ваш сотрудник может облегчить компании проведение таких встреч (и тем самым повысить вероятность их проведения), взяв на себя логистику, подготовку (забронировать помещение, принести печенье) и администрирование (провести встречу, распространить списки соглашений). Ваш сотрудник узна́ет много нового во время встречи, но не должен (и не обязан) комментировать документ, который подвергается анализу. Чтобы облегчить ревью кода на должном уровне, сотрудникам необходимо пройти обучение.
- Подготовка предварительного списка конфигураций оборудования и начальная подготовка к покупке или аренде оборудования.
- Запрос возможностей тестирования. Их проектирование и программирование требует времени. Если не внести их в бюджет и график, то они не попадут в код.
- Обсуждение возможности измерения покрытия кода и использования других инструментов поддержки разработки (таких как Purify или Bounds Checker). Успешное использование этих инструментов потребует поддержки (времени и внимания хотя бы одного члена команды программистов). Если в проекте это время (вероятно, половина рабочего времени в течение большей части времени тестирования ПО) не предусмотрено, то вы его не получите.
- Подготовка к автоматизации тестирования. В нее входит достижение договоренностей о масштабах автоматизации и уровне ее кадровой поддержки.
- Изучение ваших средств тестирования. Закажите программное обеспечение и оборудование для поддержки автоматизации. Научитесь их использовать.
- Заказ созданных внешними разработчиками тестовых наборов, если таковые существуют для вашего типа ПО. В более общем случае следует искать программное обеспечение, которое может служить оракулом, чтобы облегчить проведение крупномасштабного тестирования.

- Изучение рынка продукта и конкурентов. Станьте опытным пользователем как минимум двух приложений (помимо вашего) на этом рынке.

Разработка на основе контрактов отличается от разработки, ориентированной на рынок

Когда компания выполняет разработку программного обеспечения по контракту, в нем оговариваются обязанности сторон. В контракте может быть указан набор функций, порядок написания кода, тестирования, документирования и поддержки программы. Основная ответственность вашей компании заключается в выполнении своих обязательств по контракту. Если вы создаете то, что указано в контракте, то ваш клиент должен за это заплатить (под клиентом мы понимаем человека или компанию, оплачивающую программное обеспечение). По мере развития продукта клиент, вероятно, изменит свое мнение относительно некоторых его аспектов. Во многих проектах такие изменения подвергаются контролю, поскольку они влияют на стоимость проекта (кроме того, процесс контроля изменений может быть использован для ослабления переговорной позиции клиента).

Многие советы, содержащиеся в учебниках по программной инженерии, вполне применимы к крупным проектам по разработке заказного ПО, выполняемым по контракту.

Разработка, ориентированная на рынок, — совсем другое дело. Клиенты покупают ваш продукт только после того, как ваша компания его создаст. На протяжении всего периода разработки вас интересует, будет ли выпущенный вами продукт продаваться на целевом рынке. Если конкурент создает более привлекательный продукт, выпускает его быстрее или лучше проводит маркетинговые мероприятия, то тот факт, что ваш продукт полностью соответствует техническим требованиям, не имеет никакого значения. Маркетологи и продавцы будут требовать изменений в дизайне на протяжении всей разработки, основываясь на постоянном потоке информации о клиентах, конкурентах и ожиданиях прессы.

В проекте разработки на основе контракта вашей основной задачей может быть тестирование программного обеспечения на соответствие

спецификации. В проекте разработки, ориентированной на рынок, вы, скорее всего, будете заниматься выявлением ожиданий различных клиентов и тестированием продукта на соответствие им.

Задайте вопрос о характеристиках тестируемости

Если вы тот, кто возглавит работу по тестированию продукта, то, вероятно, вы будете первым тестировщиком, участвующим в проекте. Чем раньше вы запросите функции тестируемости, тем больше вероятность того, что программисты и руководители проекта согласятся выделить на них бюджет и внести их в график. В противном случае вы, скорее всего, их не получите.

В целом именно вы должны информировать проектную команду о потребностях вашей команды и о тех видах информации и поддержки, благодаря которым она будет работать более эффективно и результативно.

Более подробно идеи тестируемости мы рассматриваем в уроке 137.

Согласовывайте графики сборки

Ваши процессы должны соответствовать темпам обновления программного обеспечения. Как часто ваша компания выпускает новые сборки? Раз в месяц? Раз в неделю? Раз в день? Три раза в день?

Возможно, вы не сможете управлять сборками, которые поступают к вам слишком быстро. На квалификацию сборки (проверку ее соответствия основным тестам) и последующее обновление на всех компьютерах требуется время. Если сборки приходят каждый день, то можно тестировать одну сборку в течение нескольких дней, прежде чем обновлять ее.

Некоторые программисты требуют, чтобы вы сообщали о багах, имеющихся только в сегодняшней сборке. Все, что сообщается о вчерашней сборке, по их мнению, не имеет ценности, поскольку они уже могли найти и исправить этот баг. Это, конечно, удобно для них, но для вас это может стать кошмаром, поскольку вам придется каждый день останавливать все дела, чтобы обновить ПО, а затем повторить в новой сборке все, что вы уже наполовину исправили, прежде чем вы сможе-

те продолжить работу, которую делали вчера. Задача, которая должна занимать целый день, может растянуться на два-три дня, если прибавить время, уходящее на административную работу и отвлекающие факторы.

Единственное известное нам решение — согласовать график сборки. Договоритесь о том, как часто вы будете принимать новое программное обеспечение, как новая сборка будет признаваться пригодной для тестирования и как баги, обнаруженные в последних сборках, будут воссоздаваться в текущей сборке.

Узнайте, что программисты делают (и не делают) перед поставкой сборки

Одни команды разработчиков проводят обширное модульное тестирование перед выпуском новой сборки для тестировщиков. Другие так не делают. Одни команды проводят смоук-тестирование как часть процесса сборки. Другие — нет.

Команда, с которой вы работаете, делает то, что делает. Не стоит полагать, что ее участники провели или не провели определенные виды тестов или проявили особую осторожность при подготовке сборки для вас. Узнайте, как строится их процесс, и основывайте свои действия на знании того, что они делают.

Будьте готовы к сборке

Важно, чтобы тестовые среды были готовы к моменту выпуска очередной сборки. Это особенно актуально для веб-пространства. В быстро развивающемся проекте команда тестировщиков, не имеющих хорошо управляемой тестовой среды, бесполезна.

Иногда следует отказаться от тестирования сборки

Иногда сборку приходится отвергать и отказываться от ее тестирования. Для этого могут быть веские технические причины.

- Если важность этой сборки заключается в том, что она добавляет критически важную функцию, а вы обнаруживаете, что она

отсутствует в сборке или сразу не работает, то дальнейшее тестирование — пустая трата времени.

- Если ключевые функции, которые раньше работали, теперь сильно повреждены, то, скорее всего, сборка была создана с использованием неправильных файлов или файлов, которые будут быстро заменены. Баги, о которых вы сообщите в этой сборке, скорее всего, будут проигнорированы («Ах да, это был баг с плохим файлом. Данный файл мог испортить все что угодно. Можете ли вы получить этот баг снова?»). Ваш смоук-тест должен выявить этот уровень отказа. Обычно сборка отклоняется автоматически, если смоук-тест выявил проблему.
- Если вы только что получили сборку, но знаете, что через несколько часов появится другая, на которую никак не повлияет то, что вы найдете в этой, то в зависимости от стоимости квалификации и установки сборки вам, возможно, лучше проигнорировать только что полученную сборку и продолжить тестирование на старой, ожидая следующую.

Общий принцип заключается в том, что вам следует отказаться от тестирования сборки, если этот процесс существенно снизит эффективность вашей работы, не принеся ощутимой выгоды, или если ваша работа над этой сборкой будет проигнорирована.

Используйте смоук-тесты, чтобы провести квалификацию сборки

Смоук-тест (также известный как проверка работоспособности или приемка в тестирование) — это набор тестов, целью которого является проверка базовой функциональности сборки. Если сборка не проходит тест, то объявляется настолько нестабильной, что ее не стоит тестировать.

Как правило, когда новая сборка передается на тестирование, один тестировщик запускает смоук-тест (он может быть автоматизированным, ручным или комбинированным). Остальные тестировщики игнорируют новую сборку до тех пор, пока она не пройдет смоук-тестирование. Обычно в нем присутствует стандартная, основная серия тестов, а также несколько вре́менных (тестов, которые будут отменены через несколько сборок), предназначенных для поиска багов или

проверки функций, представляющих особый интерес в данной сборке.

Смоук-тестирование — общедоступный процесс. В тех компаниях, где он налажен, программистам предоставляются копии всех документов, описывающих тесты, и всего автоматизированного тестового кода, выполняющего смоук-тесты. В некоторых из этих компаний программисты проводят смоук-тестирование автоматически, как часть процесса сборки. Все понимают, что если программа не пройдет тест, то будет исключена из тестирования. Процесс проходит без сюрпризов. В таких условиях смоук-тестирование рассматривается как техническая, а не политическая задача, и большинство участников проекта (в том числе большинство или все руководители) считают его целесообразным.

Иногда правильное решение — остановить цикл тестирования и исправления и спроектировать программное обеспечение заново

Если вы продолжаете находить баги в одной и той же области, сколько бы исправлений ни было сделано, или все еще находите одну и ту же задачу пользователя запутанной, сколько бы мелких изменений в пользовательский интерфейс ни внесли, то, вероятно, пора остановить тестирование и отладку этой области. Возможно, она нуждается в переработке и повторном написании кода.

Как менеджер по тестированию, вы можете предложить это действие руководителю проекта. Вы можете использовать систему отслеживания багов, чтобы показать статистику (много отчетов, много исправлений, много новых отказов), подтверждающую ваше предложение.

Мы предлагаем сделать эту презентацию или рекомендацию руководителю проекта в частном порядке и быть готовыми к тому, что вы не сможете убедить его принять рекомендуемые вами меры. Руководитель проекта управляет проектом. Вы оказываете ему услуги, в том числе даете хорошие советы, но он может совершать свои ошибки. Поначалу его правильные решения могут показаться вам неверными.

Как и везде в этой книге, в данном случае играет роль корпоративная культура. Некоторые компании ожидают, что вы представите эти

рекомендации и связанные с ними данные публично. Мы советуем обратить внимание на тон вашего доклада в таких компаниях: вы выявляете проблему и делаете предложение, которое руководитель проекта может принять или отклонить.

Адаптируйте свои процессы к фактически используемым практикам разработки

Один консультант, выступая на недавней конференции по тестированию ПО, посоветовал слушателям отказаться от тестирования продукта, пока программисты не предоставят подробные спецификации. Это ужасный совет.

Подобные советы встречаются довольно часто, причем уже не менее 20 лет. Мы считаем, что они скорее подорвут авторитет менеджера по тестированию или приведут к его увольнению, чем позволят добиться каких-либо улучшений в процессах компании.

Отметим и отложим в сторону вопрос: *действительно ли* подходы с большим количеством документации, не допускающие поздних изменений (например, водопадный), являются единственными, которые можно считать «хорошей инженерией»? Именно это предположение (часто явно выраженное) лежит в основе такого рода советов.

На наш взгляд, ключевыми вопросами, возникающими при подобного рода предпочтениях, являются следующие.

- Должна ли ваша команда тестировщиков разрабатывать свои техники таким образом, чтобы требовать от программистов вашей компании того, чего они не делают, не хотят и в настоящее время не должны делать?
- Реально ли это?
- С каких пор вы стали руководителем проекта или вице-президентом по развитию? Если вы хотите управлять проектом, то станьте тем, кто уполномочен делать это.

Мы предлагаем принимать программистов такими, какие они есть. Продавцы и заезжие консультанты, не заинтересованные в вашем успехе, которых никак не коснутся последствия ваших неудач, — плохой авторитет в вопросе о том, какие обязанности должны выполнять программисты в вашей компании.

«Проектные документы — интересная выдумка: они полезны, но их никогда не достаточно» (Брайан Марик)

Даже в проекте, в котором делается попытка полностью описать продукт, документы по разработке (такие как спецификации и требования) будут оставлять много простора для воображения. Не боритесь с этой истиной, она фундаментальна.

Рассмотрим обработку ошибок. Мы видели оценки, согласно которым более 80 % кода в современных проектах по разработке ПО связано с обработкой ошибок и менее 20 % — с решением основных задач. При этом даже в якобы подробных спецификациях обработке ошибок отводится менее 20 %. Это означает, что 80 % кода создается программистами в процессе его написания.

При работе со спецификацией *запрашивайте* конкретную дополнительную информацию, чтобы заполнить пробелы. *Не разрабатывайте* тесты и не планируйте проект, исходя из предположения, что то, что написано на бумаге, является полным, исчерпывающим или точным.

Не просите о предоставлении того, чем не будете пользоваться

Тестировщики любят говорить: «Нет спецификации, как же я могу тестировать без нее?» Если вы требуете спецификацию, то используйте ее. Убедитесь, что руководитель проекта и составители спецификации знают, что вы ее используете, и понимают, как вы это делаете. Иначе в будущем они откажутся давать вам то, что однажды оказалось невостребованным. Они скажут: «Почему я должен вам это давать? Вы заставили меня потратить столько времени на написание спецификации, а сами так и не воспользовались ею».

Воспользуйтесь другими источниками информации

Вы не беспомощны, если никто не дает вам спецификацию. Множество других источников информации могут помочь направить ваши мысли в нужное русло.

Например, попросите сотрудников отдела маркетинга или разработки предоставить примеры использования. С их помощью можно проводить

тестирование на основе пользовательских сценариев и выявлять серьезные ошибки проектирования быстрее, чем при систематическом пролистывании списка функциональных возможностей.

Вот некоторые другие источники полезной информации, которые можно использовать для получения сведений, отсутствующих в спецификациях.

- Проект руководства пользователя (и руководство предыдущей версии).
- Литература по маркетингу продукции.
- Маркетинговые презентации, продажа концепции продукта руководству.
- Списки изменений, поставляемые с каждой новой внутренней версией программы.
- Внутренние служебные записки (например, руководителя проекта — инженерам, с описанием определений функций).
- Опубликованные руководства по стилю и стандарты пользовательского интерфейса (например, руководства, опубликованные компаниями Apple Computer и Microsoft).
- Опубликованные стандарты (например, язык Си).
- Тестовые наборы для проверки совместимости продуктов сторонних производителей.
- Опубликованные нормативные документы.
- Баг-репорты (ответы на них).
- Результаты реверс-инжиниринга программы.
- Опрос людей, таких как руководитель разработки, технический писатель, специалисты по работе с клиентами и технической поддержке, эксперты в предметной области и руководитель проекта.
- Заголовочные файлы, исходный код, определения таблиц базы данных.
- Спецификации и списки багов для всех используемых вами сторонних инструментов.
- Прототипы и лабораторные записи по прототипам.
- Беседы с сотрудниками отдела разработки, занимавшимися предыдущей версией.
- Записи звонков клиентов, пользовавшихся предыдущей версией (какие баги были обнаружены в работе?).

- Результаты тестирования удобства использования.
- Результаты бета-тестирования.
- Ziff-Davis SOS CD и другие компакт-диски технической поддержки, посвященные багам в вашем продукте и распространенным багам в вашей нише или на вашей платформе.
- Сайты, описывающие распространенные баги, такие как www.cnet.com и ссылки с www.winfiles.com, списки рассылки, новостные группы, веб-форумы и т. д., где обсуждаются баги и недостатки в руководстве по локализации вашего продукта (возможно, опубликованном для локализации продуктов на вашей платформе).
- Совместимые продукты (понять их особенности, баги и дизайн, а затем посмотреть, как ваш продукт с ними сравнится). См. списки рассылки, NEWS, BugNet и т. д.
- Точное сравнение с продуктами, которым вы подражаете.
- Справочные материалы по содержанию (например, атлас для проверки онлайн-программы по географии).

Сообщайте руководителю проекта о проблемах управления конфигурацией

Какова вероятность того, что исправление бага приведет к поломке чего-то еще в продукте или что исправленный баг появится вновь? Вероятность возникновения побочных эффектов сильно зависит от компании и проекта. Мы видели проекты, в которых побочные эффекты практически отсутствовали, и проекты, в которых они имелись в изрядном количестве.

Побочные эффекты вероятны, если код старый, сильно исправленный, сильно взаимозависимый (высокая связанность данных и/или кода) и плохо документированный. Но даже относительно чистый код иногда подвержен побочным эффектам. Одна из распространенных причин — слабое управление конфигурацией.

Если баги, которые были исправлены, появляются вновь, исправляются, опять появляются и снова исправляются — и так раз за разом, то, скорее всего, вы столкнулись с проблемой управления версиями. В таких случаях либо команда разработчиков устранит ее, либо вам придется постоянно перепроверять код, который работал в последней сборке.

Иногда проблема заключается не в управлении конфигурацией. В качестве иллюстрации Рекс Блэк рассказал нам о «тестовом подпроекте с сотнями багов, выявленными в течение четырех месяцев, где в среднем баг был повторно открыт два с половиной раза». По его словам, «проблема была не в управлении конфигурацией, а в неаккуратном исправлении, вызванном чрезмерно сложным планированием».

В любом случае обсудите эту проблему с руководителем проекта и попросите совета.

Если проблема не исчезает, укажите в отчетах о статусе проекта необходимость проведения всеобъемлющего регрессионного тестирования (и работу, которую вы для этого проводите). Если она будет повторяться в вашей компании, то заложите в бюджет будущих проектов возможность проведения широкомасштабного автоматизированного регрессионного тестирования.

Программисты подобны торнадо

Продавец инструментов тестирования пытался убедить одну компанию в необходимости создания тестовых сценариев на уровне пользовательского интерфейса с использованием метода захвата и воспроизведения. Представители компании сказали продавцу, что это не сработает, поскольку можно ожидать, что программисты будут вносить поздние изменения в пользовательский интерфейс и набор функций программы. Продавец ответил, что тестировщики должны побуждать программистов заниматься *реальным инжинирингом*. По его мнению, это означает замораживание пользовательского интерфейса и набора функций на ранних этапах проекта. Тогда тестировщики смогут выполнять свою работу.

Главная ценность такого заявления заключается в том, что оно помогает понять, что с этим продавцом (и его компанией) лучше не иметь дела. Как и в случае с уже упоминавшимся плохим советом («отказывайтесь от тестирования, если вам не предоставят спецификацию», см. урок 173), неразумно основывать свои методы на том, что команда программистов в вашей компании не будет делать.

На Среднем Западе США в домах есть подвалы. Исторически сложилось так, что подвал, помимо прочего, служил местом, где можно было укрыться в случае торнадо. Конечно, *не обязательно* тратиться на

строительство подвала. Может быть, стоит отказаться от него и объявить, что погода не предвещает торнадо. *Однако такое заявление может оказаться бесполезным, особенно когда придет следующий торнадо.*

Программисты подобны торнадо. Думайте о них как о силах природы. Они будут делать то, что делают. В разных компаниях, как и положено, программисты поступают по-разному. Выстраивайте свою работу соответствующим образом.

Тщательное планирование тестирования упрощает поздние изменения

Если поздние изменения неизбежны, то наша задача — разработать процессы тестирования, которые хорошо справляются с такими изменениями. Вот несколько предложений.

- Вместо того чтобы разрабатывать большой набор тестов перед началом тестирования, разрабатывайте их по мере необходимости. Если впоследствии изменения в продукте приведут к тому, что эти тесты устареют, то они по крайней мере будут полезны в течение некоторого времени.
- Не создавайте огромных тестовых документов, требующих больших затрат на сопровождение, таких как подробные сценарии ручного тестирования. Максимально сократите объем документации.
- Не привязывайте ручные или автоматизированные тесты к особенностям пользовательского интерфейса, за исключением тестов, предназначенных специально для пользовательского интерфейса. Даже если вы проводите сквозное тестирование, в котором он обязательно проверяется, не привязывайте тесты к его мелким деталям, поскольку они могут измениться.
- Разрабатывайте автоматизированные тесты таким образом, чтобы максимально повысить их сопровождаемость и переносимость на разные платформы (см. главу 5).
- Создайте набор типовых тестов для решения ситуаций, повторяющихся в каждой программе. Это даст возможность сэкономить время на планирование и облегчит делегирование полномочий при добавлении новой функции или ее изменении на поздних этапах проекта.
- Создайте мощный набор смоук-тестов, позволяющих быстро обнаружить основные отказы в тестируемом ПО. Если программисты вносят значительные изменения, то, вероятно, часто пересобирают

ПО и будут рады быстро присылать вам новые сборки, интегрируя изменения по несколько штук за раз и передавая их вам как можно быстрее. Смоук-тесты отсеивают плохие сборки с небольшими затратами, облегчая вам работу с частыми (например, ежедневными) сборками.

- Рассмотрите возможность использования методов экстремального программирования для разработки автоматизированных тестов (Beck, 1999; Jeffries et al., 2000). В частности, мы рекомендуем создавать общую архитектуру и дизайн серии автоматизированных тестов, а затем разрабатывать и предоставлять код итеративно, предоставлять решения в порядке, минимизирующем риски проекта (в данном случае проектом является подпроект тестирования), программировать парами и тесно сотрудничать со стейкхолдерами (другими тестировщиками, программистами и руководителем проекта), чтобы определить, что нужно делать дальше.
- Разработайте модель пользователей продукта и тех преимуществ, которые они хотят получить от него. На ее основе разработайте сложные тесты. Большинство из них не будут быстро меняться по мере реализации проекта, поскольку ориентированы на преимущества, а не на детали реализации.
- Помогайте программистам разработать большой набор модульных и других тестов с относительно простой функциональностью. Они могут выполняться всякий раз, когда код пересобирается, перед отправкой его на тестирование.

Эти предложения не так важны, как общий принцип. Проанализируйте свою технику тестирования и текущую ситуацию. Определите, какими будут ваши затраты при внесении поздних изменений в программное обеспечение, насколько это повлияет на вашу эффективность. Затем найдите способы изменить свои процессы, чтобы сократить эти затраты или распределить их на весь период разработки, а не оставлять на конец.

Возможности тестирования открываются всякий раз, когда один человек передает артефакт другому

Программы не пишутся сразу целиком, они пишутся функция за функцией и компонент за компонентом. Аналогично обстоит дело со спецификациями, документами с требованиями и руководствами: они пишутся раздел за разделом, глава за главой. Возможность для тестирования появляется в любой момент, когда любая часть продукта готова к ревью.

Таким образом, тестирование проводится на протяжении всей разработки продукта. Тестируйте то, что к этому готово, как только оно будет к этому готово.

Не существует универсальной формулы, позволяющей определить достаточный объем тестирования

Нам бы хотелось, чтобы существовал способ точно знать, что мы провели достаточное количество тестов. На самом деле таких формул было предложено множество. По нашему мнению, все они имеют существенные проблемы. Более того, мы считаем, что лучше жить в состоянии неопределенности по отношению к тестированию, чем питать ложную уверенность. Оценка достаточности объема тестирования всегда основывается на вашем опыте и знании всех факторов, влияющих на этот вопрос.

Так что не беспокойтесь о поиске формулы. Заботьтесь о том, чтобы адекватно использовать ваш опыт.

«Достаточный объем тестирования» означает «достаточное количество информации для моих клиентов, чтобы они могли принять взвешенное решение»

Поскольку тестирование — это процесс сбора информации, его можно прекратить, когда собрано достаточно информации. Можно прекратить тестирование после того, как будут обнаружены все баги, но, чтобы узнать о том, что все они найдены, потребуется бесконечное количество тестов, поэтому такой вариант не подходит. Вместо этого вам следует остановиться в тот момент, когда вы обоснованно считаете, что вероятность существования в продукте других важных ненайденных проблем мала.

На принятие решения о том, что качество тестирования является приемлемым (достаточно низкая вероятность необнаружения серьезных багов), влияют несколько факторов.

- Вы знаете, какие проблемы было бы важно найти, если бы они существовали.
- Вы знаете, как в различных частях продукта могут проявляться важные проблемы.

- Вы изучили продукт в той степени и таким образом, которые соответствуют этим рискам.
- В вашей стратегии тестирования использовались настолько разные техники, что это позволило избежать туннельного зрения.
- Вы использовали все доступные ресурсы для тестирования.
- Вы выполнили все стандарты процесса тестирования, которых ожидают от вас клиенты.
- Вы выразили свою стратегию тестирования, результаты тестирования и оценки качества настолько определенно, насколько смогли.

Если все это вы делали умело и добросовестно, то любой неприятный баг, возникающий после выпуска, скорее всего, будет вызван одной из трех причин.

- Вы не настолько хорошо понимали динамику риска, как вам казалось. Теперь вы знаете больше.
- Вы допустили ошибку при тестировании. В следующий раз вы сделаете свою работу лучше.
- Ваша оценка риска была правильной. Но руководство решило пойти на риск. Такое случается.

По мере накопления опыта работы с линейкой продуктов ваша способность определять, какого объема информации достаточно, будет возрастать. Пропустить баг при тестировании — не грех. Грех — быть небрежным, легкомысленным или не хотеть учиться на своем опыте.

Никогда не планируйте только два цикла тестирования

Некоторые лаборатории закладывают в бюджет два цикла тестирования. По их мнению, в первом цикле будут выявлены все баги, во втором — проверены все их исправления. Пока не добавляются новые тесты, продукт никак не меняется, кроме исправления багов, и все эти исправления идеальны, все будет работать прекрасно. В реальном же мире вам, вероятно, придется тестировать продукт гораздо чаще, чем дважды.

- По мере того как вы будете больше узнавать о продукте (во второй раз вы знаете больше, чем в первый), вы будете придумывать новые, более подходящие тесты и находить новые баги. Если вы работаете по двухцикловой модели, то ваше стремление к созданию новых тестов во втором цикле тестирования будет сильно затруднено.

- Даже если все программисты попытаются исправить все баги, найденные в первом цикле тестирования, вероятность того, что они исправят их все и не создадут новые, ничтожно мала (если только не были найдены всего один или два бага).
- Часто можно обнаружить, что некоторые тесты не удается выполнить с первого раза из-за дефектов, которые не позволяют даже попытаться их провести. Это блокирующие дефекты. Заблокированные тесты будут корректно запущены впервые значительно позже. А что если они обнаружат баги?

Раз за разом мы наблюдаем, как проекты планируют два цикла тестирования, а затем срываются. Первоначальные ожидания были нереальными, поэтому изменения графика были неизбежны, но каждое изменение рассматривалось командой тестировщиков как отказ и задержка.

Создавая график для набора задач, оцените количество времени, необходимое для каждой из них

Ваша работа будет состоять из большого количества задач. Составьте их список. (См. Kaner 1996b и любую книгу по управлению проектами, в которой рассматривается оценка иерархической структуры работ.) Одни задачи вы можете знать только в общих чертах, другие можно разбить на достаточно мелкие части. Насколько это возможно, создавайте отдельные пункты для всего, что занимает больше одного дня. Оцените (примерно), сколько времени займет решение каждой задачи, суммируйте это время, добавьте 25 % (или больше либо меньше, в зависимости от компании) на встречи, обучение и другую работу, не связанную с проектом, и используйте итоговый результат в качестве оценки общего времени.

Этот метод кажется более простым, чем он есть на самом деле. Составление списка задач — нетривиальная задача, так как очень легко пропустить задачи или недооценить их объем.

Попробуйте использовать другие методы для получения приблизительных оценок.

- Если вы уже выполняли другие подобные проекты, то оцените, сколько времени займет этот, опираясь на прошлый опыт.
- Если у вас есть представление о величине и сложности программы и основанная на данных вашей текущей компании модель, в которой

величина и сложность связаны с продолжительностью тестирования, то примените эту модель для получения оценки.

- Если у вас есть представление об основных рисках, связанных с проектом, то оцените, что потребуется (время и задачи) для тестирования этих конкретных рисков и для легкого тестирования продукта в целом.
- На ваши оценки будут влиять некоторые другие факторы. Например, если вы знаете, что какие-то программисты особенно искусны в работе с приложениями данного типа, то их код, вероятно, потребует меньшего объема тестирования. Если же эти отдельные программисты допускают больше багов, чем обычно, то проект, вероятно, потребует дополнительного тестирования. Если пользовательская документация уже написана или имеется четкая и подробная информация от пользователей, то тестирование, вероятно, будет более быстрым и легким.

Используя подобные методы, вы сможете оценить, сколько тестировщиков потребуется на любом этапе проекта. Чем дальше движется проект (чем больше информации вам доступно), тем точнее должны быть оценки.

Время выполнения задачи должен определять исполнитель

Руководители часто недооценивают задачи, которые делегируют другим. Если вы хотите получить обоснованную оценку, то узнайте ее у человека, которому придется работать с задачей, и требуйте от него соблюдения сроков. Если оценка тестировщика намного больше вашей, то не стоит начинать с попыток заставить его изменить свою оценку. Вместо этого постарайтесь понять, каков, по его мнению, объем задачи, что еще делает этот тестировщик и какие другие факторы привели его к оценке, которая кажется завышенной. Это может стать отличной возможностью помочь вашему тестировщику работать более эффективно. Возможно, вы сможете переформулировать для него задачу, а возможно, вам придется пересмотреть свою оценку. Запугивание тестировщика с целью изменить оценку приведет к ее уменьшению, но задача все равно займет столько же времени. Ваш график догонит реальность позднее.

В связи с этим мы решили, что вам будет интересно прочитать содержательный ответ Пэта Макги:

> «“Время выполнения задачи должен определять исполнитель…” Мне не очень нравится эта практика. Мне кажется, что многие люди не уделяют достаточно внимания тому, как они делают работу, позволяющую давать адекватные оценки. Мне приходилось сталкиваться с ситуациями, когда программист рассчитывал, что на некую задачу уйдет две недели, руководитель говорил, что ее выполнение не должно занять больше двух дней, а на самом деле ушло всего полтора дня. При этом проект прошел ревью с первой попытки. Но я также видел обратные примеры, когда решение задачи занимало гораздо больше времени. Я думаю, что правильную оценку дает тот, кто более взвешенно оценивает время, необходимое для выполнения работы. Иногда это руководитель. Иногда это исполнитель. Иногда ни тот ни другой. Независимо от этого, я считаю, что ваше первоначальное утверждение слишком упрощено до такой степени, что становится бесполезным».

Разница между нашими подходами заключается в том, что в утверждении Пэта оценку дает человек, обладающий более полными знаниями. Мы же берем оценку человека, который столкнется с последствиями, если она окажется неверной (и, соответственно, у которого есть стимул получить наиболее точную оценку). Оба подхода кажутся нам адекватными и куда более действенными, чем распространенный *метод оценки на основе принятия желаемого за действительное*.

Не существует правильного соотношения количества тестировщиков и других разработчиков

Нас часто спрашивают, какое соотношение количества тестировщиков и других разработчиков является правильным. Мы считаем, что это некорректный вопрос (Kaner et al., 2000; Hendrickson, 2001a).

Во-первых, два человека могут иметь в виду совершенно разные вещи, когда говорят, что их соотношение один к одному. Поэтому компании различаются тем, кого они учитывают, когда начинают считать, и на какие задачи ориентируются при сравнении тестировщика с другими специалистами. Это делает соотношение между компаниями несопоставимым.

Во-вторых, такое соотношение привлекает внимание к себе, а не к выполняемой работе. Предположим, что в последнем проекте программисты потратили 16 человеко-месяцев на разработку и написание кода,

а тестировщики — 24 человеко-месяца на поиск багов. Соотношение (24 к 16 или 3 к 2) является точным, но неинформативным. Измените соотношение нового и стороннего кода, количество кода из предыдущих проектов, который может быть использован повторно, соотношение обязанностей по устранению багов в отчетах таким образом, чтобы тестировщики в этот раз устраняли баги больше (или меньше) времени, чем в прошлый раз, или многие другие переменные — и соотношение из прошлого проекта окажется неверным для текущего. Вместо того чтобы вообще говорить о соотношении, поговорите о том, какую работу вам нужно сделать и сколько людей для этого потребуется.

Меняйте задачи или переводите людей с задач, с которыми они не справляются

Тестировщик, который отлично справляется с тестированием качества данных, может быть просто перегружен, когда пытается спланировать эффективное тестирование продукта в широком диапазоне конфигураций. У разных тестировщиков разные сильные стороны. Поощряйте тестировщика рисковать и выкладываться по максимуму; если это возможно, то найдите ему наставника для такой работы, но при этом следите за успехами этого тестировщика. Если он справляется с задачей чрезвычайно медленно или продолжает писать неэффективные баг-репорты, значит, работа ему не подходит (слишком сложная, слишком техническая в той области, в которой он не разбирается, слишком скучная или связанная с личным конфликтом с разработчиком). Не заставляйте тестировщиков выполнять неподходящие задачи. В идеале можно найти другого тестировщика, который согласится взять на себя эту задачу в обмен на отказ от другой своей задачи. Возможности для обмена задачами часто находятся, если их искать.

Меняйте тестировщиков при работе над функциями

Не заставляйте тестировщика работать с одним и тем же набором функций с самого начала тестирования проекта и до его окончания. Во-первых, большинству тестировщиков будет скучно. Во-вторых, со временем этот тестировщик специализируется настолько, что станет для вас менее ценным, чем менее опытный специалист общего профиля. В-третьих, если ваш специалист уйдет, то в вашей команде образуется огромный провал в знаниях. В-четвертых, и это самое важное,

два тестировщика будут анализировать одну и ту же функцию поразному. Они начнут с разных теорий ошибок, создадут разные тесты и найдут разные дефекты. Когда один из тестировщиков начинает чувствовать уверенность в качестве важной области программы, переместите его из этой области. Новый тестировщик, назначенный на этот участок, найдет дефекты, которые предыдущий и не думал проверять.

Фрэн Маккейн предлагает оценить ваш контекст:

> «Здесь следует соблюдать осторожность. Некоторые наборы функций достаточно велики, и изучение области, позволяющее эффективно ее протестировать, требует времени. Слишком ранняя ротация приводит к тому, что тестировщики работают на стабильно низком уровне вклада».

Мы согласны, что специализация важна, но задаемся вопросом, как управлять возникающими при этом рисками. Возможно, один из способов снизить их — проводить тестирование в парах, когда специалист работает в паре с другими тестировщиками из вашей команды.

Попробуйте тестирование в парах

Пара тестировщиков, работающих вместе, часто может работать не хуже (или даже лучше), чем опытный охотник за багами. Пары могут быть стабильными (два человека регулярно работают вместе), а могут быть гораздо более изменчивыми, как, например, в *экстремальном программировании*. В этом случае тестировщик, отвечающий за определенную область, будет искать краткосрочных партнеров, обладающих полезными навыками или знаниями, позволяющими оказывать воздействие на ту или иную часть этой области.

Парное тестирование отличается от многих других видов парной работы тем, что тестирование — это *деятельность по генерации идей*, а не по реализации планов. Тестирование — это эвристический поиск в неограниченном и многомерном пространстве. Работа в паре побуждает каждого тестировщика объяснять свои идеи и реагировать на них. Когда один тестировщик должен изложить свои мысли другому, этот простой процесс, как представляется, позволяет больше сфокусироваться на идеях, что естественным образом рождает новые идеи. Мы считаем, что при правильном подходе данный процесс приведет к появлению большого количества рабочих идей, которые лягут в основу тестов.

Мы настоятельно рекомендуем парам тестировщиков согласовывать устав до начала сеанса работы. Чтобы создать устав, они могут провести 5–10 минут вдали от компьютера (возможно, стоя перед флипчартом), размышляя о том, в каком направлении хотят двигаться в течение ближайших часа или двух. Они могут сосредоточиться, например, на рисках, которые необходимо исследовать, на прогнозируемых багах, которые предстоит найти, на функциях, которые нужно протестировать, или на необходимых инструментах. Это общее руководство, а не подробный список тестовых случаев. Тестировщики могут отклоняться от плана, чтобы изучить новую возможность (например, отследить подозрительное поведение, которое они только что заметили, но которое связано с другой функцией). Однако когда они делают паузу после такого отклонения, то должны обратиться к своему уставу, чтобы решить, что делать дальше. Мы ожидаем, что без устава пары иногда будут терять концентрацию.

Мы также считаем, что парное тестирование помогает обоим тестировщикам сосредоточиться на задаче, облегчает воспроизведение и анализ багов, а также позволяет одному тестировщику не отвлекаться от работы, пока другой занимается какой-то иной задачей. Мы также подозреваем, что напарники будут реже отвлекать друг друга по незначительным вопросам. К тому же программировать в паре более весело.

Мы полагаем, что работа тестировщиков в парах не менее эффективна, чем работа в одиночку. Они успевают сделать больше за меньшее время. *Попробуйте тестирование в парах.* Посмотрите, как оно работает в вашей компании. Возможно, вы будете использовать эту технику только для решения каких-то конкретных классов задач или для организации работы определенной группы людей.

Назначьте в проект охотника за багами

Охотник за багами — это опытный мотивированный тестировщик-исследователь. Вот несколько способов, позволяющих использовать навыки такого специалиста.

- Проведите начальное исследовательское тестирование подозрительной области, чтобы разработать идеи более детальных воздействий, которые могут быть выполнены менее опытными тестировщиками.

- Исследуйте область, которая предположительно имеет низкий уровень риска. Сможет ли охотник быстро найти баги, которые приведут к переоценке риска?
- Устраняйте основные области, в которых, как представляется, могут возникать баги, не поддающиеся воспроизведению.
- Найдите критические ошибки, которые убедят руководителя проекта перенести (преждевременную) дату выпуска.

Заведите устав сессий тестирования, особенно исследовательского

Прежде чем сесть за компьютер, тестировщик (или пара) должен четко представлять, что его ждет, чем он намерен заниматься в течение следующих 60–90 минут. Мы называем это *уставом сессии*. Мы все еще экспериментируем с этим подходом, но, похоже, он хорошо работает.

Преимущество устава в том, что он помогает тестировщику сосредоточиться на задаче и не отвлекаться на непродуктивные дела. Устав не должен ограничивать тестировщика. Если он видит что-то подозрительное или ему в голову приходит гениальная идея, то он может ее реализовать. Но когда он делает перерыв, то сверяется со своим уставом сессии и возвращается к нему, если нет ничего более привлекательного или более срочного.

Мы считаем, что особенно полезно разрабатывать четкий устав, когда два тестировщика работают в паре.

Чтобы разработать устав, тестировщик может опираться на подробный план проекта и выбрать задачу, которая займет день или меньше, и/или создать страницу флипчарта, на которой будет описана работа в этой одной сессии или в ближайших нескольких.

В уставе сессии может быть указано, что нужно тестировать, какие инструменты использовать, какие тактики применять, какие риски при этом возникают, какие баги следует искать, какие документы изучать, какие результаты желательны и т. д.

Тестируйте сессиями

Сессия — это защищенный отрезок времени от 60 до 90 минут. Во время сессии тестировщик проводит целенаправленное тестирование.

Если вы менеджер по тестированию, то защищаете целостность сессии — фактически вешаете на кабинет тестировщика большой знак «Не беспокоить», который все (и вы в том числе) должны соблюдать, если только не требуется срочно решить серьезную проблему.

Тестировщики, которые не находят способов защитить свое время, часто вынуждены работать урывками, поскольку им приходится отвечать на вопросы и ходить на совещания.

Используйте журналы активности, чтобы выявить то, что мешает тестировщикам работать

Если вы не считаете, что вам нужно защищать свое рабочее время от постоянных прерываний, то ведите журнал активности в течение недели или двух. Укажите в нем каждый телефонный звонок (и время, когда вы на него отвечали), каждую паузу, отведенную на чтение электронных писем и ответы на них, каждое совещание и каждый раз, когда кто-то заглядывал к вам в кабинет с вопросом, шуткой или по иной надобности. Каким был самый длинный непрерывный отрезок времени, который вы смогли посвятить планированию или тестированию? Удалось ли вам найти *любые* приемлемые отрезки времени в обычное рабочее время или же вы приходили на работу очень рано или задерживались допоздна, чтобы какое-то время поработать непрерывно? (Более подробное обсуждение фрагментации времени см. в Weinberg, 1992, с. 284).

Если вы менеджер по тестированию, работающий с тестировщиком, у которого, возможно, проблемы с продуктивностью или который, чтобы справиться с нагрузкой, использует чрезмерное количество сверхурочных часов, то журнал активности — полезный способ помочь и тестировщику, и вам. Ему журнал может помочь сфокусироваться и расставить приоритеты в работе, а вам — понять, что нужно сделать, чтобы помочь сотруднику. Мы не советуем использовать этот журнал в качестве дисциплинарного инструмента или средства формальной оценки работы. По этим вопросам обратитесь в отдел кадров за консультацией, который будет работать в конкретной компании. Мы хотим сказать, что иногда люди бывают перегружены. Если кто-то из ваших подчиненных выглядит перегруженным, то обратите внимание на инструменты, которые помогут такому работнику сконцентрироваться на поставленных задачах, эффективно использовать время и избавить-

ся от задач, которые невозможно выполнить. Журнал активности — один из таких инструментов.

Регулярные отчеты о состоянии — мощный инструмент

Истинная сила команды тестировщиков — коммуникативная, а не административная. Вы *убеждаете* людей предоставить необходимые вам ресурсы, выполнить нужную работу и пересмотреть решение о выпуске неудовлетворительных продуктов для клиентов.

Отчеты о состоянии — полезный инструмент, помогающий донести информацию. Вот несколько советов, как максимально использовать их возможности.

- Всегда пишите в нейтральном тоне, основываясь на фактах. Воздержитесь от восклицательных знаков, заглавных букв и юмора.
- Избегайте выделения отдельных лиц. Говорите о поставках, багах и сроках, но концентрируйтесь на объекте, а не на человеке, связанном с ним.
- Примите стандартный формат, общий для всех проектов. Не настраивайте себя на то, что руководитель (проект которого столкнулся с проблемами) будет жаловаться на то, что вы относитесь к его проекту как-то по-особенному, поскольку он вам не нравится.
- Распространяйте отчет по стандартному графику. На некоторых ранних этапах проекта отчет можно публиковать один раз в две недели. В дальнейшем это можно делать один раз в неделю. Ближе к концу проекта отчет может быть ежедневным. Установите для всех проектов эквивалентные графики.
- Тщательно подбирайте материал для отчета. Он должен быть лаконичным, содержащим большое количество информации на нескольких страницах.
- Разошлите отчет людям, не входящим в команду проекта. Отправьте его начальнику руководителя проекта и, возможно, начальнику начальника, а также всем группам стейкхолдеров в компании. Руководитель проекта может возразить, что вы рассылаете отчет слишком многим людям. Мы предлагаем два варианта ответа: «Мы всегда рассылаем подобные отчеты этим людям» и «Если этот отчет не подходит для г-жи X, то просто попросите ее сказать нам, что

она не хочет его получать. Мы с радостью прекратим его отправку по ее просьбе[1]».

Нет никого опаснее, чем вице-президент со статистикой

Сообщая о состоянии дел (или проводя другие измерения), следите за тем, что вы считаете и с кем делитесь этими данными. Топ-менеджеры, в частности, склонны принимать решения, используя данные, которые им непонятны. По определению, измерение — это тонкий срез общей картины. Любое измерение, сводящее картину к небольшому набору цифр, является грубым упрощением. Если вы понимаете контекст измерения, то у вас есть надежда на продуктивное использование измерений. Высшее руководство часто не имеет такого контекста.

Мы призываем вас экспериментировать с измерениями. Мы используем их для получения информации о том, как идет проект и как обеспечивается качество продукта. Руководители беспокоят нас, поскольку, по нашему опыту, используют метрики не для обучения, а в первую очередь для установления контроля над тем, чего они не понимают.

Значит ли это, что вы никогда не должны предоставлять цифры руководителям? Во многих компаниях это нереально. Но вы можете быть

---

[1] Несколько рецензентов отметили, что в их компаниях этот последний совет может быть опасным и контрпродуктивным. Мы понимаем, что в некоторых компаниях информационный поток более ограничен и руководители имеют больше прав на контроль потока информации. С другой стороны, эффективность вашей работы будет ограничена, если вы не сможете донести до стейкхолдеров проекта информацию о выявленных вами проблемах. Возможно, вам помогут следующие предложения.

- Если вы только внедряете идею отчетов о состоянии тестирования в свою команду проекта, то рассылайте отчеты количеству человек, максимально возможному в рамках культурного контекста вашей компании. Используйте неформальные каналы для того, чтобы о существовании этих отчетов узнала более широкая группа людей, и пусть они попросят руководителя проекта внести их в список рассылки. В итоге если ваши отчеты будут полезны, то вы сформируете широкий список, который будете использовать во многих проектах. В какой-то момент такая рассылка станет корпоративной практикой. Именно в этот момент вы можете позволить себе сказать руководителю проекта: «Мы всегда рассылаем эти отчеты этим людям».
- Если вы оказываете услуги только руководителю проекта и не имеете более широких обязанностей перед компанией, то у вас нет причин высылать свои отчеты другим людям. С другой стороны, если вы можете быть привлечены к ответственности за то, что не сообщили о ключевых проблемах другим сотрудникам компании, то вам необходим открытый канал для представления отчетов о состоянии проекта этим людям.

осторожны в отношении того, что предоставляете, можете предвидеть недоразумения и заранее предупреждать о проблемах, а также можете отказаться от предоставления определенных цифр. Приведем несколько примеров.

- Не стоит врываться в кабинет руководителя с жалобой на то, что средний программист тратит на исправление бага 1,4 недели, а Джо — 5,3. После того как вы добровольно предоставите данные об индивидуальных показателях продуктивности из системы отслеживания багов, от вас потребуют много подобной информации.
- Когда вы отправляете служебную записку, в которой говорится, что проект будет выпущен не ранее чем через пять недель, поскольку открытых багов 200, а программисты исправляют в среднем 40 в неделю, поставьте рядом с этой цифрой предупреждающее примечание. Вы можете указать, что эта цифра имеет смысл, если багов множество; но когда открытых багов мало, то на количество времени, необходимого до выпуска, больше влияют многие другие факторы проекта, чем количество багов.
- Когда вас просят предоставить статистику по отдельным сотрудникам (например, количество багов на одного тестировщика), отвечайте отказом. Если вас спросят, то объясните, что как только вы начнете использовать систему отслеживания багов для сбора данных по персоналу, характер базы изменится. Процесс регистрации багов станет более политизированным, более состязательным и менее точным.

Будьте осторожны, измеряя прогресс проекта с точки зрения количества багов

Количество багов — излюбленная система показателей прогресса проекта. К сожалению, она неадекватна и часто вводит в заблуждение.

Количество багов может показать, что проект далек от даты выпуска. Если команда проекта исправляет в среднем 40 багов в неделю, а до выхода продукта должно быть исправлено 400 открытых багов, то у вас есть аргументы, позволяющие доказать, что продукт не будет готов к выпуску в ближайший месяц.

Количество багов не позволяет показать, что качество проекта близко к качеству, характерному для окончательной версии ПО. Если к желаемому времени окончания проекта количество открытых багов невелико, то означает ли это, что продукт стал более стабильным или что

команда тестировщиков тратит слишком много времени на написание отчетов, выполнение регрессионных тестов (которые редко находят новые ошибки), демонстрацию продукта на выставках и другие действия, не направленные на поиск новых багов? Мы не можем определить это по количеству багов.

Особенно нас не впечатляют статистические модели скорости появления багов (сколько багов будет найдено за единицу времени) как средства управления проектами, поскольку мы не видим оснований полагать, что предположения, лежащие в основе вероятностных моделей, каким-либо образом соответствуют реалиям проекта. В работе Симмондса (Simmonds, 2000) дается четкое и ясное описание предположений одной из таких моделей.

Хоффман (Hoffman, 2000) приводит несколько ярких примеров фиаско, связанных с такими показателями. См. также работы Остина (Austin, 1996) и Кейнера (Kaner, 2000a).

Чем больше независимых метрик покрытия вы используете, тем больше знаете

Вы можете измерить, какая часть продукта была протестирована, по многим критериям: баги, требования, код, конфигурации, история изменений, разработчик, тестировщик, данные и т. д. Никакого одного критерия не будет достаточно.

Например, одним из часто рекомендуемых показателей является процент строк кода (строк плюс ветвей или других вариаций этого показателя), которые были выполнены во время тестирования.

Как и в случае с количеством багов, показатель количества строк кода позволяет показать, насколько ваш продукт далек от соответствия требованиям или завершенности. Если вы протестировали только 10 % строк кода, то не должны быть полностью уверены в надежности программы. Если покрытие строк приближается к 100 %, это не говорит о том, что продукт скоро будет выпущен. Это говорит лишь о том, что по этому показателю он уже не так далек от выпуска. (Дополнительные комментарии к этому показателю см. в Marick, 1999).

Предположим, вы разрабатываете продукт для массового рынка. Вы решили протестировать его на десятке различных версий Windows, под десятком версий браузеров, с использованием десяти различных соеди-

нений (модемы с разной скоростью, Ethernet и др.) и так далее для нескольких других переменных конфигурации. Можно свести все к сотне стандартных конфигураций, с которыми должна быть совместима программа, и протестировать их на всех ста.

- После того как продукт будет протестирован на первых нескольких конфигурациях, запуск тестов на других может привести к тому, что ни одна дополнительная строка кода и ни одна дополнительная ветвь в программе не будет выполнена. Считая количество строк кода мерилом близости работы, мы можем отказаться от конфигурационного тестирования. В отношении продукта массового спроса такое решение, скорее всего, окажется губительным.
- Если существует 100 стандартных конфигураций и вы протестировали 30 из них, то у вас есть метрика покрытия, которая не зависит от строк кода, но все равно очень важна. Тестирование конфигурации выполнено на 30 %.

Метрика покрытия состоит из совокупности возможных тестов заданного типа, подмножества тестов, которые вы собираетесь выполнить, и другого подмножества, которое вы действительно выполнили. Покрытие можно измерить как количество выполненных тестов по сравнению с тем, сколько тестов планировалось выполнить, или как количество выполненных тестов по сравнению с общим количеством тестов, представляющих интерес. Любая из этих метрик может быть полезной. Ни одна из них не измеряет ничего, кроме вашего прогресса по одному критерию, будь то выполненные строки или опробованные конфигурации, проверка потоков данных, проверка классических рисков (вспомните деление на ноль — баг заключается в отсутствии кода, предотвращающего это деление; такие отсутствия невидимы для счетчиков строк) или другие критерии. Можно придумать сотни возможных критериев, по одному на каждый класс возможных отказов (Kaner, 1995a, 2000a)[1].

---

[1] Даже это чрезмерное упрощение. Как пояснил один из наших рецензентов, Ноэль Найман, «это подразумевает, что все тесты имеют одну единичную стоимость, что, скорее всего, не так. Точно так же, как баги имеют разную степень серьезности, тесты имеют разную ценность и разное время выполнения. Мы можем выполнить 80 % наших тестов и получить только 20 % необходимых нам тестовых часов. Возможно, мы выполнили 80 % тестов, а осталось выполнить 50 % наиболее важных. Любая метрика, которая игнорирует это, чтобы дать некое значение “полного покрытия”, дает ложную информацию и, как правило, ложную надежду».

Используйте сбалансированную систему показателей, чтобы сообщать о состоянии по нескольким критериям

Сбалансированные системы показателей часто используются для оценки состояния бизнеса (Kaplan, Norton, 1996; но см. Austin, 1996). Более простые показатели неэффективны.

- Например, если вы ориентируетесь на квартальную прибыль, то, скорее всего, будете стимулировать поведение, направленное на максимизацию квартальной прибыли. В краткосрочной перспективе вы можете ее увеличить, уволив сотрудников отдела исследований и разработок. В течение двух лет компания будет получать большие прибыли, а затем обанкротится, поскольку ей нечего будет продавать.
- Вместо этого, возможно, следует подсчитывать количество выданных новых патентов, но этот путь подразумевает чрезмерные инвестиции в исследования и недостаточные — в производство товарной продукции.

Несколько различных цифр, казалось бы, информативно отображают состояние бизнеса, но любая из них, рассматриваемая в отдельности, чревата пагубными побочными эффектами. Ключевым моментом измерения является побуждение людей к улучшению показателей, и поэтому люди, желающие хорошо работать, будут пытаться найти способы их повысить. Методы этих людей могут не совпадать с тем, что имел в виду разработчик метрики. Например, увольнение научных сотрудников ради увеличения краткосрочной прибыли может не соответствовать ожиданиям человека, стремящегося повысить квартальную прибыль. Но это может быть честным ответом на требование увеличить данный показатель.

Возможное решение проблемы побочных эффектов измерений — использование нескольких чисел, имеющих уравновешивающие друг друга побочные эффекты. Например, система показателей, отображающая краткосрочную прибыль, количество выданных патентов и текучесть кадров, позволит быстро выявить непорядок в области НИОКР на фоне высокой краткосрочной прибыли.

Хорошо сбалансированная система показателей может быть достоверным показателем состояния бизнеса, даже если ни один из ее компонентов, рассматриваемый отдельно, не является достоверным или безопасным.

Мы считаем, что те же самые рассуждения применимы к прогрессу проекта и прогрессу тестирования. Отчетность о прогрессе (или его отсутствии) можно рассматривать по нескольким различным критериям.

- Какой объем продукта был протестирован?
- Насколько полно выполнено запланированное тестирование?
- Сколько проблем было обнаружено и сколько из них еще не решены?
- Насколько мы уверены в качестве тестирования (например, уровень нашей уверенности должен быть низким, если бета-тестировщики находят вопиющие ошибки, пропущенные тестировщиками)?
- Какой объем работ по тестированию блокируется из-за неисправленных дефектов, отсутствия оборудования или непринятых решений?

В лучших отчетах о состоянии, которые мы видели, используются показатели каждого из этих типов. Информация, которую руководство получает благодаря им, не укладывается в одно тривиальное число. Но информационная схема может быть представлена достаточно просто, что позволяет принимать решения.

Рекомендуемая структура еженедельного отчета о состоянии

Представьте отчет о состоянии в виде документа, состоящего из четырех страниц.

На первой перечислены ключевые вопросы.

- *Необходимые решения.* (Например, как расставить приоритеты; какое оборудование мы тестируем; брать ли нам нового сотрудника?)
- *Необходимые исправления багов.* (Все, что мешает работе, должно быть исправлено в приоритетном порядке.)
- *Ожидаемые поставки* (например, обещанные документы, оборудование, функции и инструменты) и обещанные сроки поставки. Они попадают в список за некоторое время до наступления срока их выполнения и остаются в нем после того, как срок их выполнения истек. Удалите все, что было поставлено. Это список того, чего не хватает, а не пополняемый перечень работ. Выделите все, что задерживает работу, например доступ к обещанному оборудованию или завершение и доставку списка функций.

- *Неожиданные проблемы.* (Например, сообщите о снижении эффективности тестирования в той или иной области из-за текучести кадров; ключевой инструмент работает не так, как ожидалось; люди нуждаются в обучении и т. д.)

На второй странице описывается прогресс вашей команды в выполнении запланированных задач. Например, вы можете перечислить области тестирования в вашем графике, сколько вы выделили на каждую из них (например, две недели), сколько вы выполнили по каждой из них (например, 10 %) и сколько времени потратили на эту работу. В данном примере, если вы потратили более 10 % от двух недель, предусмотренных планом, вы отстаете от графика по этому направлению. Просматривая весь список, вы создаете общее впечатление о прогрессе (или его отсутствии) в сравнении с планом и графиком.

На третьей странице представлена статистика баг-репортов. Мы поместили информацию о номерах багов на внутреннюю страницу по той же причине, по которой газеты помещают на таких страницах рубрики о спорте. Эти рубрики — самая читаемая часть газеты. Если вы поместите данные о багах на внутреннюю страницу, то на первой странице можете дать новости, которые, по вашему мнению, важно сообщить, а на других страницах — разместить рекламу (или привести другие материалы, о которых вы хотите сказать).

На последней странице перечислены баги, которые были отложены на этой неделе. Список, вероятно, содержит только номер бага, строку с резюме (или названием) и оценку серьезности бага, присвоенную тестировщиком. Читатель, желающий получить более подробную информацию, может найти ее сам.

Информационная панель — еще одно полезное средство для отображения состояния проекта

Информационная панель представляет собой диаграмму, нарисованную на большой доске в конференц-зале, открытом для всех членов команды проекта и всех, кто имеет отношение к нему. На ней наглядно показано состояние проекта. (Дальнейшее обсуждение и иллюстрации см. в Bach, 1999b.)

На типичной информационной панели представлены несколько областей работы, по одной в строке. Даже на большой доске не поместится много областей, если вы хотите, чтобы ваши слова были достаточно боль-

шими, чтобы их можно было прочитать. По каждой области на доске отображаются текущие трудозатраты; уровни покрытия: запланированный и достигнутый на данный момент; оценка качества, присвоенная тестировщиком (по шкале: высокое, среднее и низкое). На доске также остается место для того, чтобы тестировщик сделал несколько ключевых комментариев, подтверждающих его оценку.

Типичная информационная панель обновляется еженедельно (в начале проекта), а затем ежедневно или даже дважды в день по мере приближения даты окончания проекта. Она должна быть достаточно актуальной, чтобы менеджеры, пришедшие к вам с кратким визитом, захотели на нее взглянуть.

Информационные панели проекта в разных компаниях могут различаться. Например, некоторые компании хотят иметь конкретную информацию по каждой области либо хотят, чтобы одна или две строки были переменными, зарезервированными для статуса проблемы или функции недели.

Точная компоновка не имеет значения. Главное — наглядно показать состояние проекта.

Пример панели показан в табл. 8.1.

Таблица 8.1. Пример информационной панели тестирования

Информационная панель тестирования				**Обновлено: 21 февраля Сборок: 38**
Область	**Трудозатраты**	**Покрытие**	**Качество**	**Комментарии**
Файл/редактирование	Высокие	1	☺	
Просмотр	Низкие	1+	😐	1345, 1363, 1401
Вставка	Низкие	2	J	
Формат	Низкие	2+	K	Нарушена автоматизация
Инструменты	Заблокировано	1	L	Аварийные отказы: 1406, 1407
Слайд-шоу	Низкие	2	L	Утечка памяти при анимации
Онлайн-справка	Заблокировано	0		Новые файлы не доставляются

Продолжение ⇩

Таблица 8.1 *(продолжение)*

Информационная панель тестирования				**Обновлено: 21 февраля Сборок: 38**
Область	**Трудозатраты**	**Покрытие**	**Качество**	**Комментарии**
Клипарт	Нет	1	К	Нужна помощь в тестировании…
Преобразователи	Нет	1	😐	Нужна помощь в тестировании…
Установка	Начало 17 марта	0		
Совместимость	Начало 17 марта	0		Запланировано лабораторное время
Общий GUI	Низкие	3	☺	

Отчеты о пройденных этапах полезны, когда эти этапы четко определены

Одни компании реализуют свои проекты, соблюдая этапы, и проводят тщательную оценку состояния на каждом из них. Другие — нет.

Если ваша компания хочет получить оценку продукта на каждом этапе, то необходимо понять, по каким показателям вы оцениваете продукт. Как компания определяет этап? С какими аспектами этого определения вы должны сравнивать продукт? Например, если в определении этапа говорится, что код 50 % функций написан, вы должны указать, верно это или нет. Как вы это узнаете?

Если в вашей компании нет стандартного определения этапа, то вы не сможете однозначно сказать, готова ли, например, бета-версия продукта тестирования к тестированию. Ваши суждения в отсутствие стандарта компании, скорее всего, вызовут больше политических распрей, чем понимания.

Если вас просят оценить прогресс относительно какого-либо этапа, а определения данного этапа нет, то мы предлагаем сначала договориться с соответствующими руководителями об определении, а затем на его основе дать оценку. Если эти люди обращаются к вам за советом, как определить этап, предложите им думать о нем как о завершении итерации проекта. Каковы критерии завершения этой итерации? (Или каковы критерии входа в следующую итерацию?) Лоуренс и Джонсон на-

писали об этом в своей очень полезной работе (Lawrence, Johnson, 1998), которая содержит подробную информацию и доступна бесплатно[1].

Не ставьте свою подпись в знак одобрения выпуска продукта

Пусть руководитель проекта или команда решают, когда выпускать продукт. Ваша задача — убедиться, что они имеют полную информацию (какую только может предоставить тестировщик), позволяющую принять это решение. Сообщайте о своих выводах честно, прямо, точно и всем стейкхолдерам. Если они примут решение, которое вы бы не приняли, это будет их выбор.

Поставьте свою подпись, чтобы показать, что вы протестировали продукт и остались довольны

Если кто-то настаивает на том, чтобы вы подписали форму выпуска продукта (которая разрешает руководителю проекта запустить продукт в производство или выпустить ПО в промышленную эксплуатацию), то дайте понять, что ваша подпись означает, что вы провели адекватное (по вашему мнению) тестирование, достигли согласованного уровня тестирования или что ваша команда прекрасно справилась с поставленной задачей за отведенное время. Вы можете дать такое пояснение в служебной записке (адресованной руководителю или команде проекта) или написать короткий комментарий под своей подписью на формах выпуска.

Если вы пишете отчет о выпуске, то описывайте проделанную работу по тестированию и ее результаты, а не ваше мнение о продукте

Команде тестировщиков очень трудно оценить общее качество продукта или его пригодность для рынка, на котором он будет продаваться.

---

[1] На сайте www.coyotevalley.com/plc/builder.htm приведен полезный пример подробных определений нескольких этапов проекта. Мы не утверждаем, что эти определения правильные, и не считаем, что Лоуренс и Джонсон считают их правильными. Напротив, они показывают тип критериев, с помощью которых можно давать определение этапа, и служат хорошей отправной точкой для компании, находящейся в процессе создания собственных определений.

У вас нет соответствующих данных. Зато у вас есть баг-репорты и ваши попытки найти баги (тесты, которые продукт прошел). Опишите то, что вам известно.

Приведите список неисправленных багов в финальной версии отчета о выпуске

Если вы готовите (или помогаете готовить) финальный отчет о выпуске, то в него следует внести список неисправленных багов продукта. В списке также нужно указать отклоненные проблемы дизайна, которые вы считаете важными. Разошлите этот список заранее, чтобы финальный отчет не содержал сюрпризов.

В полезном отчете о выпуске будут перечислены десять худших моментов, которые могут заметить критики

Если вы выпускаете программное обеспечение для перепродажи, то представьте, что пишете отчет о выпуске, в котором перечислены десять основных проблем, которые мог выявить критически настроенный обозреватель журнала. Если проблемы окажутся достаточно серьезными, то отдел маркетинговых коммуникаций вашей компании (или другой маркетинговый персонал) задержит выпуск. Возможно, это самая полезная оценка качества, которую вы можете дать. (Обратите внимание, что суть этого подхода не в том, чтобы дать справедливую оценку, а в том, чтобы попытаться представить проект с позиции критика.)

Глава 9

Управление командой тестировщиков

Данная глава посвящена проблемам управления командой тестировщиков. Разница между нею и главой 8 заключается в том, что глава 8 посвящена проблемам одного проекта. Как работать с другими членами команды разработчиков и другими стейкхолдерами, чтобы помочь выпустить нужный продукт в срок и в рамках бюджета? В этой главе мы рассмотрим группу людей, которые работают вместе проект за проектом, год за годом.

Выбрать один из наших любимых уроков было непросто, поскольку у нас троих достаточно большой опыт управления. У каждого из нас был собственный бизнес, и мы могли бы написать очень длинный раздел общего характера, посвященный управлению техническим персоналом. Чтобы не делать этого, мы предлагаем несколько ссылок на работы Вайнберга (Weinberg, 1992, 1997a, 1997b, 1997c, 1998 и 2001), Хамфри (Humphrey, 1997), Деминга (Deming, 1986), Демарко (DeMarco, 1997), Демарко и Листера (DeMarco, Lister, 1999), Брукса (Brooks, 1995), Константина (Constantine, 1995), Блэка (Black, 1999), Друкера (Drucker, 1985) и Вигерса (Wiegers, 1996). В них можно найти несколько общих уроков, особенно на тему подбора персонала. Мы добавили их в книгу потому, что получаем очень много вопросов о найме и поиске работы, и поэтому решили, что ни одно обсуждение на тему того, как управлять командой тестировщиков, не будет полным без рассмотрения этих уроков.

Посредственность — самоисполняющееся пророчество

Если вы:

- устраняете творческий подход, дегуманизируя свои процессы;
- ставите себе цель сделать персонал взаимозаменяемыми шестеренками;
- стандартизируете работу своих сотрудников, чтобы каждый мог делать именно то, что умеет делать, в том же виде, в каком он это делает;
- оцениваете своих сотрудников с помощью цифр, которые не отражают их творческий потенциал, убедительность, рассудительность или чувствительность к межличностным отношениям;
- отстраняете сотрудников от результатов их работы, говорите им, что их функция заключается лишь в том, чтобы сообщать о багах, а кто-то другой будет решать, что исправить, а эти сотрудники не будут в этом участвовать, и не позволяете им высказываться по поводу своих багов на тематических совещаниях,

то не ждите, что они будут:

- проявлять активность, когда проект больше всего нуждается в них;
- высказываться по принципиальным вопросам;
- с радостью работать по ночам, когда проекту нужен дополнительный толчок;
- разрабатывать отличные тесты для сложных, но критических ошибок;
- не спать ночами, подыскивая слова, с помощью которых можно объяснить критическую проблему руководителю, просто не понимающему, о чем речь...

И не ждите, что они проявят лояльность к вам или к вашей компании, когда вам это будет нужно больше всего.

Если вы заявите, что в вашей компании не должно быть героев, то не получите ни одного[1].

[1] Подробнее о том, что мы понимаем под героями, читайте в работах Симса и Манца (Sims, Manz, 1996), а также Лебоу (Lebow, 1990).

Относитесь к своим сотрудникам как к руководителям

Питера Друкера часто называют основателем современной теории менеджмента. В одной из своих замечательных книг, *The Effective Executive*[1], он учит, что руководитель — это тот, кто управляет стоимостью своего времени и влияет на работоспособность организации. Большинство работников умственного труда — руководители. Друкер отмечает, что руководителям всегда поручают больше дел, чем они могут выполнить. Эффективные руководители выбирают подмножество задач, с которыми справятся, и полностью абстрагируются от многих других. Неэффективные — пытаются делать все, не добиваются успеха и не особенно преуспевают во многом из того, что пытались сделать. У разных руководителей разные сильные стороны и интересы. Если дать двум руководителям одинаковую работу (не задачи, а постоянную работу), то они будут выполнять ее совершенно по-разному.

В понимании Друкера, большинство тестировщиков — это руководители. Управляйте ими, исходя из этого. Не руководите ими так, будто они заводские рабочие, и не тратьте свое время на то, чтобы переделать их работу, превратив ее в эквивалент работ, выполняемых на заводе. Вместо этого примите тот факт, что у них разные сильные стороны и интересы, и управляйте ими, исходя из него.

Говоря об этом, мы имеем в виду *не только* старших тестировщиков. Формируя команды тестировщиков или управляя ими, мы нанимаем специалистов разного уровня опыта и подготовки. Новые тестировщики требуют повышенного внимания со стороны руководства. Нам приходится оплачивать обучение младших сотрудников. Однако наша цель — подготовить руководителей, и мы с первого дня работы проявляем к ним уважение как к обучающимся, мыслящим личностям. Мы ожидаем, что они быстро возьмут на себя ответственность, проявят энтузиазм, завоюют авторитет и будут совершенствовать свои навыки. Если им нужна менее сложная работа, то они могут найти ее в другом месте.

Читайте баг-репорты своих сотрудников

Некоторые руководители или супервизоры по тестированию читают каждый баг-репорт, поданный их сотрудниками (это легко сделать,

[1] *Друкер П.* Эффективный руководитель.

если в команде семь человек, и гораздо сложнее — если их 70). Это помогает понять состояние продукта, сильные стороны сотрудников и их моральный дух, а также коммуникационные и межличностные проблемы, с которыми они сталкиваются.

Вот некоторые вопросы относительно баг-репортов, которые помогут оценить качество работы тестировщика.

- Хорошо ли написаны отчеты?
- Просто ли излагается в них проблема?
- Есть ли в них участки, которые требуют последующего тестирования?
- Тестирование, в ходе которого были обнаружены баги, кажется рутинным или глубоко продуманным?
- Трудно ли было найти баг? Находится ли баг в обычно стабильной части приложения и если да, то о чем это говорит: об упорстве тестировщика, сообщившего о баге, или о его удаче?
- Каков тон отчета?
- Понял ли отчет программист? Какие комментарии он добавил к нему? Какое впечатление создается о взаимодействии программиста и тестировщика: они сотрудничают или говорят каждый о своем?

Оценивайте своих сотрудников как руководителей

Количество багов — это в лучшем случае бесполезный показатель эффективности работы вашего персонала. Он вводит в заблуждение и демотивирует, а также заставляет думать, что вы знаете гораздо больше, чем есть на самом деле. В худшем случае из-за этого показателя ошибающиеся сотрудники производят впечатление хороших, а лучшие выглядят некомпетентными.

Количество багов сохранилось как показатель, поскольку менеджеры и руководители заявляют, что им нужен какой-то способ оценки эффективности работы их сотрудников, и не знают, что еще можно использовать, кроме количества багов. Мы не можем предложить простые метрики, но можем указать, что вы можете сделать, чтобы оценить работу вашего сотрудника:

- читайте его баг-репорты;
- читайте его код;
- читайте его тестовую документацию;

- собирайте комментарии программистов и других стейкхолдеров, работающих с ним.

Рассмотрите следующие ситуации.

- В какие ссоры он ввязывается и почему?
- Насколько хорошо он соблюдает сроки?
- Насколько хорошо он выполняет собственные обещания?
- Какие проблемы он упускает из виду?
- Какие виды помощи он оказывал другим тестировщикам и программистам, чтобы сделать их работу более эффективной или продуктивной?
- Приобретает ли он новые навыки? Насколько хорошо передает свои навыки другим тестировщикам?
- По каким вопросам он высказывался в вашей компании? Как это отражается на его деловых суждениях и личной этике?

Информация, полученная в результате этой работы, даст вам комплексное представление о работе вашего сотрудника и о состоянии проектов, над которыми он работает. Эта информация не сводится к чему-то простому, но вы можете использовать ее для построения оценочной карты. Представьте, что по каждой области (баг-репорты, код и т. д.) тестировщик получает оценку от 0 до 5 баллов. В получившейся карте будут представлены его сильные и слабые стороны (как вы их воспринимаете). Кроме того, аналогичная многомерная информация поможет вам разработать план обучения для каждого сотрудника.

Многие компании используют самооценку как часть формального процесса оценки продуктивности персонала и оплаты труда (сотрудник заполняет анкету, чтобы оценить собственную работу). Этот метод может быть очень полезным, но некоторые руководители слишком полагаются на такие оценки и не уделяют достаточно времени тому, чтобы узнать больше о реальной работе сотрудника. Еще одна проблема, которая нас беспокоит, заключается в том, что формальные оценки проводятся редко: раз в полгода или год. Мы призываем вас активно следить за работой своих сотрудников и делать это регулярно. (Более подробную информацию см. в работе Деминга (Deming, 1986, с. 102 и далее).) Обратите внимание, что мы *не* предлагаем опираться на микроменеджмент: изучать, что делают ваши сотрудники, и обучать их на основе этих знаний — деятельность, разительно отличающаяся от раздачи указаний, когда и как им надлежит выполнять свою работу (Drucker, 1985).

Если вы действительно хотите знать, что происходит, выполняйте тестирование вместе с сотрудниками

Будучи руководителем, вы наверняка не раз слышали, что «хороший руководитель делегирует». Конечно, вам придется делегировать большую часть работы своей команде, и, скорее всего, вы не сможете сыграть главную роль в каком-либо проекте. Однако если вы не будете работать в проектах, то со временем потеряете многие свои технические навыки, не будете видеть реальных проблем, с которыми сталкиваются ваши тестировщики, и вам будет сложнее оценивать качество работы сотрудников. Участие сразу в нескольких проектах часто нецелесообразно, но иногда при тщательной расстановке приоритетов можно заметить, что активное участие хотя бы в одном проекте за раз достаточно важно, чтобы посвятить этому занятию некоторое время.

Вы потеряете доверие сотрудников, если не сможете даже установить или запустить тестируемое программное обеспечение. Мы настоятельно рекомендуем вам также воспроизвести несколько багов, о которых сообщается в отчетах. Ознакомьтесь с продуктом настолько, чтобы ваши сотрудники могли достаточно глубоко обсуждать его с вами.

Не ждите, что люди будут эффективно управлять несколькими проектами

Если у вас больше проектов, чем тестировщиков, то у вас возникнет соблазн поручить каждому тестировщику несколько проектов. Остерегайтесь двух проблем.

- Некоторые люди соглашаются на несколько заданий, но в течение недели (или месяца) будут работать только над одним из них. Остальные отходят на второй план.
- Тестировщик, активно работающий над всеми порученными проектами, тратит слишком много времени на решение организационных вопросов. В итоге вы получаете человека, который работает над несколькими проектами, ходит на множество совещаний, тратит много времени на то, чтобы наверстать упущенное по каждому проекту (перечитывает свои записи, проверяет последние баги в базе данных, просматривает отчеты о новых функциях, читает много электронных писем и т. д.), но мало что делает по каждому из проектов (DeMarco, Lister, 1999).

Повышайте уровень компетентности персонала в предметной области

Эффективность работы ваших сотрудников резко возрастет, когда они узнают больше о внешних фактах, ограничивающих разработку продукта, о том, как люди используют (или будут использовать) продукты, подобные этому, какие проблемы будут для них важны, как конкуренты решают их проблемы и что уже написано об использовании данного типа продукта.

Этот опыт можно накопить несколькими способами.

- Читайте журналы и книги, которые были написаны для людей, подобных вашим клиентам.
- Посещайте занятия (желательно, чтобы их проводили не сотрудники вашей компании), на которых клиентов обучают тому, как использовать продукты, подобные разрабатываемому.
- Посещайте курсы, на которых преподается предмет, лежащий в основе продукта. Например, если вы продаете программу для использования в сфере продажи недвижимости, то пройдите обучение, чтобы стать брокером по недвижимости, ипотечным брокером или оценщиком.
- Работайте на объектах ваших клиентов. Это можно делать на условиях временной аренды (например, ваша компания может отправить вас к важному заказчику на несколько недель для установки программного обеспечения и обучения работе с ним) или частичной занятости (убедитесь, что у вас есть разрешение вашей компании).
- Продавайте свое программное обеспечение или ПО конкурентов. Например, работая менеджером по разработке программного обеспечения в компании, производящей потребительское ПО, один из нас, Кейнер, узнал больше о рынке сбыта своего продукта, продавая программное обеспечение в одном из ведущих розничных магазинов в течение шести месяцев по субботам. И издатель, и розничный продавец знали о его целях и поощряли его в этой работе.
- Отвечайте на звонки по горячей линии технической поддержки компании в течение нескольких часов в неделю. На начальном этапе вам потребуется помощь сотрудника службы технической поддержки. Со временем у вас все получится, и вы многому научитесь (если будете относиться к работе серьезно, задавать вопросы и изучать некоторые из услышанных жалоб).

Повышайте квалификацию сотрудников отдела тестирования в области соответствующих технологий

По мере усложнения программно-аппаратных сред все больше проблем в работе приложений будет связано с проблемами взаимодействия вашего приложения с удаленными серверами или другим программным и аппаратным обеспечением, неподконтрольными разработчикам приложения. В книге Нгуена (Nguyen, 2000) тестировщики веб-приложений могут познакомиться с тестированием подобных проблем взаимодействия. Чтобы выполнять такое тестирование успешно, необходимо много знать о другом аппаратном и программном обеспечении. (Нгуен приводит полезный вводный материал.)

Кроме того, ваша команда станет работать более эффективно, если некоторые сотрудники будут разбираться в технологиях программистов. Например, будет полезно, если некоторые члены вашей команды тестировщиков много знают об используемой программистами библиотеке компонентов и о том, как писать программы для тестирования компонентов через API.

Активно работайте над повышением квалификации

Вы не можете просто сказать: «Сегодня я освою построение графиков причинно-следственных связей». Если стоит освоить методы поиска багов и составления отчетов или методы оценки багов, то, скорее всего, придется изучить несколько подзадач и несколько раз отработать их на практике. Свои навыки совершенствовать трудно, поскольку технологии, конкурентный рынок, средства разработки меняются, а продукты весьма разнообразны. Однако несмотря на сложность повышения квалификации, вы должны делать это, чтобы оставаться в курсе событий. В дальнейшем это позволит вам внести весомый вклад в развитие вашей компании.

Просматривайте журналы технической поддержки

Чтобы получить представление о том, какие проблемы могут возникнуть после того, как программа будет установлена и начнет работать, просмотрите записи жалоб клиентов вашей компании (записи телефонных разговоров, письма и электронные сообщения, направленные в вашу компанию клиентами). Спросите себя, что можно было сделать, чтобы

снизить вероятность того или иного типа звонков. Спросите себя, с помощью каких тестов можно было бы выявить подобные типовые проблемы, если таковые имеются в тестируемом программном обеспечении.

Помогайте новым тестировщикам успешно работать

Принимая кого-то в свою команду, выполните несколько действий, чтобы первые недели его работы были успешными.

- Выберите место (офис или кабинет) для тестировщика и обустройте его (предоставив два компьютера и другое оборудование и программное обеспечение, а также необходимую документацию по компании и проекту) до его прихода.
- Запланируйте как минимум один день встреч с новым сотрудником. Проведите его по отделу, познакомьте со всеми, с кем ему предстоит работать, расскажите о своих надеждах и ожиданиях.
- Назначьте наставника. Он знакомит нового тестировщика с текущей обстановкой, готов ответить на вопросы, наблюдает за его работой и вносит свои предложения, несколько раз угощает тестировщика обедом и в целом помогает ему. Наставник не имеет надзорных полномочий над новым тестировщиком и ничего не сообщает вам о нем, если не возникает серьезных проблем. Некоторые компании назначают наставников не из отдела нового сотрудника (наставником тестировщика может быть программист или, например, маркетолог). Преимущество такого подхода состоит в том, что он способствует развитию коммуникации между отделами. Недостатком является то, что наставник может быть не в состоянии обучить тестировщика техническим нюансам его работы.

Попросите новых тестировщиков проверить документацию на соответствие программному обеспечению

Новый тестировщик — это человек, который впервые участвует в данном проекте. Он может иметь или не иметь опыт тестирования других программ.

Один из способов ознакомить нового тестировщика с вашим продуктом — попросить его проверить руководство и онлайн-справку на соответствие поведению программы. К концу такого тестирования он познакомится со всеми частями программы.

Проверка руководства и онлайн-справки — очень важная задача. Пользователи обращаются к ним, когда им нужна достоверная информация о программе и когда у них возникают проблемы. Кроме того, в случае продукта, предназначенного для массового рынка, в документации, вероятно, содержится гарантия того, что продукт будет работать в соответствии с описанием. Этот тип гарантии, основанный на фактическом письменном заявлении производителя, нельзя отменить (исключить) с помощью типовых договоров, используемых компаниями-разработчиками программного обеспечения. Все, что сказано в документации, может быть использовано в качестве доказательства, подтверждающего утверждение о дефектности продукта (Kaner, 1995b; Kaner, Pels, 1998).

Тестировщик должен проверить каждое утверждение о фактах (все, что может быть доказано как истинное или ложное) с помощью тестов программы. Кроме того, тестировщик должен проверить все выводы, которые, по его мнению, могут быть сделаны на основании документации. Это позволяет провести хорошую подготовку нового тестировщика и дает возможность найти важные баги (Pettichord, 2001b).

Ознакомьте новых тестировщиков с продуктом, используя положительное тестирование

После того как тестировщик ознакомился с пользовательской документацией и понял, что должен делать продукт, попросите его попробовать использовать продукт в простых, но реальных условиях. Пусть он установит продукт с нуля, возможно, на новую систему. Попросите его перечислить действия, обычно выполняемые с помощью продукта, а затем попросите выполнить ряд простых действий в нем.

Цель ваших просьб — помочь тестировщику понять, что хорошего есть в продукте или его концепции. Понимая это, он будет более информированным и более эффективным критиком, когда начнет искать неудачи, а не успехи.

Попросите начинающих тестировщиков редактировать старые баг-репорты, прежде чем писать новые

Начинающий тестировщик — новичок в тестировании, а не просто новичок в вашем проекте. Он должен освоить основные навыки тестирования, а не только приемы тестирования данного продукта.

Один из способов обучения нового тестировщика — попросить его повторно протестировать старые баги и написать отчет (который вы проверите) с другой формулировкой, которая была бы более эффективной. Возможно, это потребует изменения формулировки и последующего тестирования (поиск более серьезных проблем, связанных с заявленной проблемой, или поиск более широкого спектра обстоятельств, при которых она проявляется). Дайте тестировщику обратную связь по поводу тона, ясности и креативности последующего тестирования.

Когда вы убедитесь, что новый тестировщик может составлять приемлемые отчеты, разрешите ему описывать новые баги. Пока тестировщик не заслужит вашего одобрения, возможно, не стоит разрешать ему писать новые отчеты.

Попросите новых тестировщиков повторно протестировать старые баги, прежде чем поручать выявление новых

Полезный способ ознакомления новых тестировщиков с багами в базе данных — повторное тестирование старых багов. Мы используем три класса задач. Каждый из них может помочь обучить тестировщика тому, как устроен продукт, как он может не работать, как его можно тестировать и что находится в базе данных.

- **Репликация открытых на данный момент багов.** В отношении каждого проверенного бага тестировщик сообщает, что тот все еще может быть воспроизведен. В этом отчете тестировщик указывает дату, идентификатор текущей сборки программы, а также конфигурацию аппаратного и программного обеспечения, на котором проводилось тестирование. В качестве альтернативы тестировщик может сообщить, что баг не воспроизводится, и указать ту же дату, сборку и конфигурацию.
- **Повторное тестирование исправленных багов.** В отношении выбранных багов тестировщик проверяет, действительно ли исправлены баги, о которых заявлено, что они исправлены. Эти баги, вероятно, закрыты. В большинстве компаний запись в протокол не вносится, если только баг не появился вновь. В этом случае (в зависимости от политики компании) тестировщик снова открывает и обновляет существующий баг-репорт или подает новый, содержащий перекрестные ссылки на старый, но описывающий текущее нарушение.

- **Повторное тестирование исправленных, но незакрытых багов.** Баг возник вновь — исправленный, отложенный, невоспроизводимый или отклоненный каким-либо другим способом. Тестировщик, сообщивший о баге, проведет его повторное тестирование и, возможно, закроет этот баг. Попросите нового тестировщика тоже заняться исследованием бага. По окончании этого процесса пусть он поговорит с человеком, который сообщил о баге и повторно его протестировал. Обсудите с новым тестировщиком его подходы к устранению багов, выясните, на чем основывается его мнение о том, сколько времени и усилий следует потратить на повторное тестирование данного бага. Когда вы убедитесь, что новый тестировщик обладает достаточным уровнем знаний и навыков для устранения неисправностей, позвольте ему закрывать отчеты.

Не ставьте начинающих тестировщиков на почти готовые проекты

Если вам нужно добавить персонал в проект, который близок к завершению, то не назначайте новичка. Введите в проект того, кто много знает о тестировании. Опытный тестировщик уже знает, как читать тестовые матрицы, диаграммы границ и остальную обычную документацию тестирования (или может быстро этому научиться). Все, что вам нужно сказать ему, — это что тестировать, а не как проводить каждый тест. Ему потребуется время на изучение продукта, но он сможет работать эффективно, не отнимая много вашего времени и времени ваших сотрудников.

А вот новичку требуется гораздо больше инструкций по проведению тестирования. Некоторые команды тратят огромное количество времени на написание пошаговых инструкций по тестированию, предназначенных для новичков. Мы считаем, что этим командам лучше потратить это время на поиск багов и их исправление. Мы также считаем, что новички, использующие такие инструкции, скорее всего, пропустят много багов, поскольку не знают, какой спектр дефектов следует искать, и никакой простой набор инструкций не сможет помочь обрести необходимые навыки наблюдения и интерпретации.

Нанимайте новичков, когда у вас есть время на их обучение или задачи, на подготовку и делегирование которых у вас уходит очень мало времени.

Не нанимайте новичков, если вы не можете уделить внимание их обучению. Не стройте свою деятельность по разработке и документированию тестов таким образом, чтобы старшие тестировщики тратили много своего времени на посредственную работу новичков.

Моральное состояние сотрудников — важный актив

Как говорил Наполеон, «моральное и физическое соотносятся как три к одному».

Сотрудники, которые верят в то, что их работа важна, что они могут выполнить поставленные задачи, если приложат усилия, и что их достижения будут оценены по достоинству, могут добиться значительных успехов. Вы, как руководитель, самый важный хранитель морального духа своих сотрудников.

- Относитесь к ним вежливо и уважительно.
- Обращайте внимание на их работу.
- Хвалите за хорошую работу, энтузиазм и честные усилия.
- Если ваши сотрудники работают допоздна, то работайте допоздна сами. Не каждую ночь, не каждый выходной день, но достаточно часто, чтобы сотрудники чувствовали, что вы разделяете с ними нагрузку.
- По возможности поручайте людям те задачи и проекты, которые им интересны. Поощряйте их за выражение своих интересов и учитывайте их.
- Если тестировщик не справляется с заданием, то назначьте помощника, проведите инструктаж или, если необходимо, замените этого тестировщика (и переведите его на другую задачу).
- Создайте возможности для обучения. Покажите, что вы цените развитие навыков и профессионализма.
- Относитесь к сотрудникам справедливо и требуйте, чтобы другие относились к ним так же.
- Никогда не вводите своих сотрудников в заблуждение. Если вы не знаете ответа на вопрос, то так и скажите. Если вы делитесь догадками о чем-либо, то уточните, что это именно догадка, а не утверждение, основанное на знании.
- Никогда не кричите на сотрудников. Никогда не пользуйтесь своей властью, чтобы запугивать других.

- Избегайте публичной критики сотрудника, но указывайте на ошибки и проблемы в частном порядке.
- Не сплетничайте о членах своего коллектива с другими людьми.
- Никогда не приглашайте сотрудника на свидание и будьте крайне осторожны, предлагая или принимая личные одолжения или подарки (даже небольшие). Отдавайте больше, чем получаете.

Не позволяйте себе становиться объектом злоупотреблений

Руководители проектов иногда выдвигают необоснованные требования. Вам не обязательно делать то, что сделать невозможно. Многие руководители постоянно пытаются повысить продуктивность каждого сотрудника на дополнительные 10 или 20 %, независимо от его текущей загруженности. Не позволяйте себе испытывать чувство вины, если не можете сделать невозможное или если отказываетесь изнурять себя и свою семью, прилагая невозможные усилия.

Разработка программного обеспечения — это не всегда работа с 9:00 до 17:00. Вы должны поддерживать свою команду, и иногда это может потребовать сверхурочной работы в течение нескольких дней или недель. Иногда руководителю проекта требуется ваша помощь, чтобы вернуть проект в нужное русло или довести его до конца. Конечно, постарайтесь помочь. Но выбирайте свой темп и принимайте на себя только выполнимые обязательства. Будьте добровольцем, а не жертвой[1].

Не нужно обещать невозможного. Не нужно лгать. Не нужно скрывать проблемы. Фактически суть вашей работы как тестировщика заключается в том, чтобы выявлять проблемы, а *не скрывать* их. Вам не нужно и никогда не следует идти против собственных убеждений.

Ваша сила заключается в том, что вы можете свободно говорить то, что есть, людям, которые должны это услышать. Если вы поставите под угрозу свою репутацию честного человека, то ослабите одну из основных опор своей силы.

Некоторые работодатели неадекватны и жестоки, и вы не можете их изменить. Если работа невыносима, то разошлите резюме и уходите, когда найдете лучшую работу.

---

[1] Не позволяйте себе попасть в «гонку на выживание». Не тратьте много времени на проект, который заведомо обречен. Это никому не поможет и постепенно причинит вам эмоциональный ущерб. См. работу Юрдона (Yourdon, 1997).

Не заставляйте персонал работать сверхурочно

Многие команды тестировщиков постоянно работают сверхурочно. В некоторых компаниях тестировщики работают по гораздо более экстремальному графику, чем разработчики.

Хроническая сверхурочная работа приводит к выгоранию сотрудников, высокой текучести кадров, цинизму, политиканству, неэффективной и неаккуратной работе. Мы призываем вас защитить своих сотрудников от этого.

Многие проекты нуждаются в коротких всплесках дополнительной помощи в критические моменты. Это нормальная часть бизнеса. И, говоря о сверхурочной работе, мы имеем в виду совсем не это. Помните, что у каждого сотрудника есть свои ограничения. У некоторых из них есть семейные или другие обязательства, которые не слишком поддаются изменениям. Смиритесь с реальным положением дел.

Некоторые компании или руководители хронически сокращают сроки выполнения проектов и ожидают, что сотрудники восполнят разницу.

- Не составляйте графики, исходя из предположения, что ваши сотрудники будут уделять работе по восемь часов в день. Они не будут. Они не смогут. Сотрудники ходят на совещания и тренинги, пишут отчеты о состоянии и служебные записки, заполняют всевозможные формы, поступающие из отдела кадров вашей компании. Если вам повезет, то шесть из восьми часов в день сотрудники будут посвящать выполнению своей официальной задачи, а все остальное время — заниматься другими делами, которые они должны делать.
- Не соглашайтесь на графики, которые заведомо невозможно соблюсти. Вместо этого как можно точнее оцените, сколько времени займут те или иные задачи. Когда вам предлагают график, который короче вашего списка задач, спросите, какие из них вы должны решить в первую очередь, а какие — в оставшееся время. Если руководитель настаивает на выполнении всех задач, то спросите, какие из них можно выполнить менее тщательно. И попросите больше сотрудников. Иногда приходится сталкиваться с неадекватным руководителем, который настаивает на том, чтобы вы согласились сделать все в невозможно короткий срок. Возможно, вы и произнесете эти слова. Однако помните, что они не волшебные. Реальность пройдется по

вашему проекту и вашим сотрудникам паровым катком, какие бы обещания вы ни давали. В условиях, когда руководители не желают прислушиваться к голосу разума, управляйте своими сотрудниками адекватно, иначе вы их потеряете.

Еще одной причиной хронических сверхурочных является руководитель, который поощряет людей за то, что они находятся на месте («на работе»). Тот, кто «работает» 80 часов в неделю, может считаться более преданным своему делу и более достойным, чем тот, кто работает 45 часов в неделю, но при этом делает больше и лучше. Ваша задача — найти способ сделать результаты работы более эффективного сотрудника заметными. Мы не можем рассказать вам в книге, как это реализовать. То, что нам удалось сделать, касалось конкретных людей (руководителей и сотрудников).

Есть еще два типа руководителей, которые могут создать проблемы со сверхурочной работой. Представители первого типа считают, что планировать разработку ПО невозможно, все графики будут срываться и поэтому лучшее, что можно сделать, — просить все, что только можно, у тех, кто выше, и давить как можно сильнее на тех, кто ниже. Управленцы другого типа, похоже, считают, что демонстрируют свою ценность для компании, уговаривая сотрудников работать как можно больше сверхурочно. Мы сталкивались с представителями обоих типов. С ними трудно спорить, поскольку их позиции, по сути, неадекватны. Хорошая новость заключается в том, что среди менеджеров и руководителей компаний-разработчиков программного обеспечения наблюдается большая текучесть кадров. Реорганизации будут происходить. Дураки приходят и уходят. Ваша задача — уменьшить влияние таких людей на персонал.

Не допускайте грубого обращения с персоналом

Иногда люди плохо обращаются с тестировщиками, повышая на них голос, называя дураками, лжецами и т. д. Если вы менеджер по тестированию и ваши сотрудники работают над сложными проектами, то, возможно, вам придется обозначить определенные границы. Будьте готовы оказать личную эмоциональную поддержку своим сотрудникам и противостоять агрессорам.

Хорошим средством обеспечения взаимной поддержки тестировщиков может стать парное тестирование.

Работая в условиях жесткого графика, некоторые сотрудники могут совершать недостойные поступки или говорить неприятные слова. При публичном взаимодействии обращайтесь с провинившимся сотрудником вежливо. В частном порядке четко объясните тестировщику, что его поведение было неприемлемым. Если такое поведение характерно для него, то выполните обычные дисциплинарные процедуры, принятые в вашей компании.

Создавайте возможности для обучения

Создайте книжный клуб, который будет собираться еженедельно или дважды в месяц. Цель — каждую неделю читать статью или главу книги и обсуждать ее на встрече (сотрудники обсуждают, вы присутствуете, но не ведете активных бесед). Пусть она будет добровольной, никто не обязан ее посещать. Однако найдите способ поощрять тех, кто регулярно посещает такие встречи и вносит в них свой вклад (например, дайте книги тем, кто показал, что будет их читать).

Организуйте учебные встречи в обеденное время (еженедельно или два раза в месяц). Иногда выступайте с докладом лично. Иногда приглашайте докладчика (возможно, менеджера по тестированию из другой компании). Возможно, кто-то согласится сделать это бесплатно при условии, что вы также побеседуете с его сотрудниками. Иногда кто-то из сотрудников может сделать презентацию, посвященную технике или сложностям тестирования определенного типа устройств либо функций приложений. Иногда вы можете выполнить пару упражнений, отрабатывая тот или иной навык.

Если ваша компания компенсирует расходы на обучение, то узнайте, какие соответствующие курсы предлагает местное учебное заведение и какие лучшие курсы, связанные с программным обеспечением, можно найти в Интернете. Аналогичным образом узнайте, какие конференции по тестированию (и другим актуальным вопросам) проводятся и какие из них наиболее применимы к проектам вашей компании. Проведите индивидуальные встречи с сотрудниками и предложите им помощь в повышении квалификации. Такие встречи — прекрасная возможность обсудить долгосрочные цели тестировщика (учитывайте их при распределении следующей группы задач). Не указывайте ему, какие курсы необходимо пройти, но обратите его внимание на курсы, которые, по вашему мнению, хорошо преподаются и имеют отношение

к текущим проектам компании или профессиональным интересам вашего сотрудника.

Как правило, отправляйте сотрудников на курсы (особенно на семинары по профессиональному обучению) и конференции парами или тройками. Они будут обсуждать друг с другом полученные знания. Попросите этих людей рассказать вам или команде тестировщиков о том, что они узнали.

Ваши решения о найме — самые важные

Нанять на работу неподходящего человека — одна из худших ошибок, которую может совершить руководитель. Скорее всего, вам придется устранять ее последствия очень долго.

Делайте все возможное, чтобы нанимать хороших сотрудников.

Нанимайте временных работников, чтобы вы могли передохнуть, пока идет набор персонала

С помощью найма временных сотрудников вы сможете уделить время поиску нужного специалиста. Даже в тех компаниях, где наем подрядчиков строго ограничен, часто отдается предпочтение именно такому варианту.

Старайтесь не принимать в команду тестировщиков людей, от которых отказались в других командах

Некоторые люди, которые плохо работают в других командах, прекрасно справятся с тестированием, но таких немного. Чем больше подобных сотрудников будет в вашей команде, тем больше пострадает ее репутация. Решительно сопротивляйтесь попыткам руководителя ввести в команду сотрудника, который не вызывает у вас доверия. Даже если этот «подарок» будет бесплатным (никто из ваших нынешних сотрудников не потеряет работу из-за этого перевода и не возникает возможности нанять кого-то еще), из-за увеличения численности персонала впоследствии вам могут отказать в просьбе нанять кого-либо (вы только что наняли человека, зачем вам еще больше людей?). Кроме того, окружающие будут ожидать, что потенциал вашей команды воз-

растет пропорционально увеличению количества сотрудников. Новому тестировщику поручают новый объем задач. Если вы получаете новую работу, а новый тестировщик ее не выполняет, то задача остальных лишь усложняется.

Ставьте планы, исходя из задач, которые нужно решить в вашей команде, и необходимых для этого навыков

Не нанимайте сотрудников, основываясь на упрощенной квалификации. Истинная квалификация тестировщика выходит далеко за рамки наличия у него ученой степени или опыта работы в течение стольких-то лет. Нанимайте людей с традиционной квалификацией или без нее, которые могут доказать вам, что способны выполнить эту работу.

Набирайте в команду людей с разным опытом

Не пытайтесь набрать штат сотрудников, имеющих степень бакалавра информатики, двухлетний опыт тестирования, опыт работы с определенным инструментом автоматизированного тестирования и т. д. Создайте группу людей, обладающих особыми сильными сторонами и очень отличающихся друг от друга. Мы наняли несколько человек, которые никогда не занимались тестированием, но имели другой значимый опыт работы в реальной жизни. Такой подход может быть очень успешным, если вы введете в группу людей с необходимыми вам навыками[1].

Мы *не говорим*, что не следует нанимать программистов. Конечно, нанимайте их. Они нужны для решения многих задач. Однако вам придется обучать их многим непрограммным областям знаний. Другими сотрудниками могут быть непрограммисты, обладающие разными знаниями, которых нет у программистов. Создайте штат сотрудников, в котором сильные стороны одних компенсируют слабые стороны других.

---

[1] Мы не всегда можем последовать его совету, но в принципе согласны с замечанием одного из наших рецензентов, Рекса Блэка: «Когда вы проводите тестирование, предполагающее использование специализированных навыков, например автоматизированное тестирование и особенно тестирование производительности, я считаю, что рациональным подходом будет убедиться, что как минимум два человека в вашей команде компетентны в этих вопросах».

Приведем примеры из нашего личного опыта успешного найма таких людей. Представьте, что к вам пришел умный человек, который на последнем месте работы был юристом. Он может проанализировать любую спецификацию, которую вы ему предоставите, и подготовлен как адвокат, директор по продажам и маркетингу (тот, кого мы наняли, обучил наших сотрудников новым методам исследования и написания баг-репортов, чтобы привлечь внимание отдела маркетинга), техник по ремонту оборудования, библиотекарь (подумайте о тестировании баз данных или других систем поиска информации). Кроме того, у него есть навыки программиста, руководителя проектов (не имеющим отношения к разработке ПО), представителя службы технической поддержки, имеющего опыт поддержки продуктов, подобных тестируемым, переводчика (особенно полезен, если ваша компания издает программное обеспечение на многих языках), секретаря (подумайте обо всей информации, которую вы собираете, храните и распространяете, а также о том, как вам и вашим сотрудникам приходится распоряжаться временем), системного администратора, разбирающегося в сетях, или пользователя тестируемого программного обеспечения. Любой (или многие) из наборов навыков, которыми обладают эти люди, может оказаться полезным для вашего проекта.

Применимость этого совета ограничена. Некоторые проекты предполагают, что тестировщик должен уметь программировать. Представьте тестирование компилятора — чтобы создавать интересные тесты, нужно уметь писать код. В ряде компаний к тестировщику, не умеющему программировать, будут относиться как к сотруднику второго сорта, как бы хорошо он ни работал. В таких случаях приходится нанимать программистов. Ваша задача — нанять программистов с разным опытом работы. По нашему опыту, это нелегко, но вполне достижимо, если вы работаете в хорошей компании, а отрасль не переживает очередной подъем и не испытывает большого дефицита квалифицированных кадров.

Следует упомянуть еще одну проблему разнообразия. Время от времени появляются статьи о расизме, сексизме и эйджизме в компьютерной индустрии, и мы с этим тоже сталкиваемся. Особые предпочтения в найме, обучении, оплате труда и продвижении по службе особенно контрпродуктивны при тестировании. Чем шире культурный и профессиональный опыт сотрудников, тем больше способов анализа программного обеспечения они применят и тем больше проблем смогут выявить. Разнообразие — важнейшее преимущество в тестировании, а не то, чего следует избегать.

Нанимайте перспективных кандидатов

Ищите сотрудников в нетрадиционных местах, особенно на ограниченном рынке труда. Так, юристы и бухгалтеры обладают сильными аналитическими способностями; как ни странно, многие из них будут заинтересованы в том, чтобы провести год (или больше), изучая методы разработки программного обеспечения. Другой пример — женщина с опытом работы старшим программистом или руководителем проекта, которая только что стала матерью-одиночкой. Она нуждается в снижении темпов работы, и на ее нынешней работе ей отказали в сокращении рабочего времени. Предлагая должность с занятостью 35 часов в неделю и гибким графиком работы, вы можете привлечь сотрудника, обладающего исключительными навыками, на год или два. Еще один пример — вышедший на пенсию руководитель, ищущий новую, возможно, менее напряженную должность.

Нанимайте в результате консенсуса

Разрешите любому штатному сотруднику вашей команды и всем, кто тесно сотрудничает с ней, провести собеседование с кандидатом на тестирование. Каждый, кто проводил такое собеседование, может наложить на него вето, если только оно не основано на половой, расовой, возрастной или иной (незаконной) предвзятости, не имеющей отношения к способности человека выполнять данную работу.

Слишком легко отмахнуться от интуиции и отговорить кого-то от вето. Тем не менее мы совершили ряд серьезных ошибок при приеме на работу, в том числе наняли людей, занимавшихся сексуальными домогательствами, проигнорировав плохо выраженную словесно, но сильную реакцию одного из наших сотрудников. Внимательно и с пониманием выслушайте опасения, высказанные одним из ваших сотрудников, и примите его вето, если только не чувствуете, что оно основано на дискриминации.

Нанимайте людей, которые любят свою работу

Ищите энтузиастов и с осторожностью относитесь к тем, у кого в прошлом были проблемы с руководителями. Особенно осторожно относитесь к тем, кто испытывает горечь по поводу своей прошлой работы.

Нанимайте честных

Ваш отдел предоставляет противоречивую информацию. Доверие клиентов — ваш самый важный актив. Их уровень доверия зависит от характера ваших сотрудников.

Во время собеседования попросите тестировщика продемонстрировать навыки, ради которых вы его нанимаете

Например, если он опытный тестировщик, то попросите его написать баг-репорт (используйте программу с открытым исходным кодом, которую написала не ваша компания, чтобы у кандидата не создалось впечатление, что вы просите бесплатную консультацию). Другой пример: если вы нанимаете архитектора по автоматизации, то попросите его проанализировать продукт и сложившуюся ситуацию и предоставить вам краткий план автоматизации (опять же рассмотрите возможность использования стороннего продукта с открытым исходным кодом). Судите о кандидате по вопросам, которые он задает, по исследованиям, которые он проводит, и по тому, как он объединяет полученную информацию в общий подход. Этот подход не обязательно должен быть идеальным, но должен быть адекватным и хорошо аргументированным. (Подробнее об этом см. в работе Демарко и Листера (DeMarco, Lister, 1999).)

На собеседовании в ходе неформальных тестов попросите тестировщика продемонстрировать навыки, которые он действительно будет использовать в работе

Некоторые команды проводят неформальное тестирование способностей, используя логические или числовые головоломки. Мы не возражаем против этих методов, но не считаем, что они настолько информативны, как думают некоторые люди.

Логические и числовые головоломки позволяют получить огромный практический эффект. Кейнер занимался ими со своей дочерью, когда ей было около 12 лет. Она довольно хорошо справилась с ними. Это не значит, что она стала умнее, и не значит, что она стала лучшим тестировщиком. Это означало, что она стала лучше решать головоломки. Этот практический эффект лежит в основе большой индустрии подготовки к тестам SAT, LSAT, GRE и другим стандартизированным вступительным

экзаменам в колледж. Практический эффект (предыдущего опыта) длится довольно долго и более выражен в тестах скорости. Они более выражены в невербальных тестах и тестах на производительность (Jensen, 1980). Человек, который отлично справляется с этими тестами, возможно, просто лучше с ними знаком. Тот, кто производит впечатление тугодума, может не иметь опыта их решения, но (по нашему опыту) быть умным человеком и отличным тестировщиком.

Тесты на скорость отбирают людей, которые мыслят быстро, но не обязательно глубоко, — ментальных кроликов. Черепахи иногда разрабатывают лучшие продукты или лучшие стратегии тестирования продуктов.

При приеме на работу просите представить примеры работ

Не все соискатели могут их представить, но многие могут дать примеры написанного ими кода, составленные ими баг-репорты, документы или иные отчеты. Люди, которые были уволены или покинули компанию на хороших условиях, часто могут получить разрешение на представление потенциальным работодателям конкретных, определенных фрагментов своей работы. Другие люди составляют отчеты по открытому программному обеспечению и/или пишут код в свободное время. Никогда не просите конфиденциальные материалы и убедитесь, что информация, которую человек вам представляет, не является конфиденциальной.

Нанимайте сразу после того, как примете решение

Быстрое прохождение документов о приеме на работу по каналам компании должно стать вашей первоочередной задачей. В случае задержки хороший кандидат согласится на другую вакансию.

Изложите свои обещания, которые вы давали при приеме на работу, в письменном виде и соблюдайте их

Споры о том, действительно ли вы кому-то что-то обещали, досаждают тому, кто считает себя обманутым, тревожат остальных сотрудников и потенциально подрывают ваш авторитет. Вы проиграете, даже если никогда не давали обещаний. Избегайте двусмысленности. И никогда, никогда не обещайте того, что не зависит от вас, например продвижения по службе.

Глава 10

Ваша карьера в области тестирования программного обеспечения

Какой может быть ваша карьера тестировщика программного обеспечения? Ответ сложен. В тестировании легко зайти в тупик; необходимо управлять своей карьерой, иначе можно остаться ни с чем. Оплата труда в сфере тестирования часто ниже, чем в других областях разработки, но может быть и выше, если вы активно развиваете свои навыки и выбираете работодателей. Текучесть кадров выше; тестировщиков увольняют или сокращают чаще, чем других столь же компетентных разработчиков. Совершенствование навыков поиска работы и ведения переговоров будет очень полезным.

Приведенные нами рекомендации по поиску работы и ведению переговоров являются типично американскими и особенно характерны для Кремниевой долины. Если вы не уверены, что наши советы применимы к вашей культуре, то посоветуйтесь с опытными коллегами.

Некоторые тестировщики считают, что будут более привлекательными кандидатами на работу и более уважаемыми, если пройдут сертификацию, и получат больше влияния в коллективе, если пройдут лицензирование. У сертификации и лицензирования есть свои плюсы и минусы. Мы рассмотрим некоторые из них.

Выберите направление карьерного роста и следуйте ему

Два распространенных направления карьерного роста в области тестирования — техническое и управленческое.

Примерами технических профессий в области тестирования (в любой из них можно пройти путь от новичка до эксперта) являются:

- программист средств автоматизации;
- архитектор по автоматизации;
- тестировщик производительности и масштабируемости;
- системный аналитик;
- аналитик и критик пользовательских интерфейсов и человеческих факторов;
- планировщик тестов;
- эксперт в предметной области;
- тестировщик «черного ящика».

Предположим, что по истечении пяти лет одна из этих специальностей стала вашей. Мы считаем, что наиболее высокую зарплату и авторитет, а также больше возможностей для трудоустройства вы получите, если станете отличным архитектором по автоматизации, а следующую по величине — если станете отличным программистом средств автоматизации. Производительность и масштабируемость в настоящее время тоже хорошо оплачиваются. По нашему мнению, самая низкая зарплата будет у тех, кто является отличным ручным тестировщиком «черного ящика» или (в зависимости от тематики) экспертом в предметной области, который больше разбирается в предмете, чем в техниках тестирования.

Здесь возникает интересный конфликт интересов. Вы можете существенно увеличить свою ценность для нынешнего работодателя, изучив достаточно широкий спектр техник тестирования и дополнив их знаниями о предметной области. Вероятно, при этом можно добиться значительно более высокой зарплаты. Однако если вы работаете в компании, которая не проявляет лояльности к своим сотрудникам, или рассчитываете сменить работу по каким-либо другим причинам, то вам лучше развивать навыки, которые ценятся больше, даже если это будет происходить в ущерб вашим знаниям предметной области и, соответственно, эффективности тестирования для данного работодателя.

Полезный принцип заключается в том, чтобы определить, какие навыки наиболее необходимы вашему работодателю (или отрасли), и приобрести их.

Примерами управленческих должностей в сфере тестирования являются:

- руководитель испытаний или супервайзер;
- менеджер по тестированию;
- директор по тестированию или директор по качеству;
- внутренний консультант;
- внешний консультант.

Вы можете найти возможности для продвижения по службе за пределами области тестирования. Тестировщики имеют более широкое и, как правило, менее глубокое представление о продукте, чем программисты. Они часто заинтересованы в продукте, его представлении и влиянии на него. Они также взаимодействуют с менеджерами и старшими менеджерами в большинстве других частей компании, связанных с программным обеспечением. В результате многие опытные супервизоры или менеджеры по тестированию получают возможность пройти обучение и перейти на другие руководящие должности, в частности на следующие:

- руководитель группы проектов или руководитель проекта;
- менеджер по технической поддержке;
- продакт-менеджер (особенно актуально для экспертов в предметной области, имеющих технические навыки);
- менеджер по документации группы;
- менеджер по поддержке продаж (в случае эксклюзивных продуктов продавцов часто сопровождает человек, который может отвечать на технические вопросы и разрабатывать вместе с клиентом прототипы применения продукта).

Еще одно направление, по которому идут некоторые тестировщики, — управление процессами, в том числе:

- специалист по метрикам ПО;
- специалист по совершенствованию процессов разработки ПО.

Мы рекомендуем подходить к управлению процессами с осторожностью. Эти должности не связаны напрямую с разработкой или при-

быльностью какого-либо продукта. Поэтому в некоторых компаниях люди, работающие на этих должностях, более уязвимы для увольнений. Кроме того, в некоторых компаниях на этих должностях работают недостаточно квалифицированные специалисты. Мы рекомендуем вам получить основательную подготовку в области математической статистики и теории измерений (прочитайте работы Цузе (Zuse, 1997) и Остина (Austin, 1996)), прежде чем соглашаться на должность специалиста по метрикам. Мы рекомендуем поработать как минимум в двух, а лучше в трех областях (например, тестирование, программирование, маркетинг), прежде чем устраиваться на должность специалиста по совершенствованию процессов.

Доходы тестировщиков могут быть выше, чем доходы программистов

Традиционно тестировщики зарабатывали меньше, чем программисты с аналогичным опытом. Так не во всех компаниях, и некоторые утверждают, что в последние несколько лет тестировщики все больше опережают программистов.

Сравнение зарплат — дело более сложное. Во время бума доткомов[1] (1998–2000 гг.) заработная плата росла по всем видам программных работ. Во многих компаниях опытные архитекторы по автоматизации тестирования ПО зарабатывали больше, чем опытные программисты. Опытные тестировщики по методу «черного ящика», которые занимались только ручным тестированием, зарабатывали не так хорошо. Во время краха доткомов (2000–2001 гг.) заработная плата падала, и мы подозреваем, что средние ставки для специалистов по ручному тестированию «черного ящика» падают быстрее.

Наше впечатление таково, что тестировщики, обладающие специальными техническими навыками, часто ценятся больше (и их труднее найти), чем тестировщики с управленческими навыками, которые, в свою очередь, ценятся больше, чем тестировщики «черных ящиков» и менее опытные автоматизаторы тестов.

---

[1] Дотком — термин, применяющийся по отношению к компании, бизнес-модель которой основывается на работе в рамках сети Интернет. Возник и получил распространение в конце 1990-х гг.

Не стесняйтесь изменить направление и заняться чем-то другим

Вы не ограничены рамками тестировании. Вы не ограничены руководящей ролью или должностью. Вы не ограничены ни одной компанией, ни одним направлением, ни одной работой.

Бизнес имеет свои циклы. Мы проходили через жесткие и открытые рынки труда, бумы и спады. Можно смело проявлять инициативу, когда рынок открыт, а работодатели отчаянно нуждаются в работниках. Однако даже при напряженной ситуации на рынке труда у подготовленных людей появляется множество возможностей.

Например, если вы хотите перейти от тестирования к программированию, то пройдите курсы. Если ваш график работы не позволяет посещать курсы в университете или программу повышения квалификации (иногда называемую профессиональным обучением), то пройдите самостоятельный курс в Интернете или изучите один из пакетов «книга и курс в одном комплекте». После того как вы приобретете базовые навыки владения соответствующим языком программирования, попросите дать вам возможность поработать *в свое свободное время* в одной из команд программистов. Если вы встретите сопротивление со стороны руководителя тестирования, то постарайтесь заверить его, что будете заниматься тестированием 40 часов в неделю. Все, что сверх, — ваше личное время, и его следует уважать. (Если вы не можете добиться таких условий и хотите заняться программированием или каким-либо новым для вас направлением, то вам, возможно, придется найти временную работу в другом месте.)

Какое бы направление вы ни выбрали, действуйте активно

Речь идет о *вашей* карьере. Ваша компания или ваш руководитель могут помочь вам в планировании карьеры, и они могут быть хороши в этом. Или нет. Если вы выбрали направление, а также имеете навыки и решимость придерживаться его, то ваша компания с большей вероятностью оплатит занятия и позволит вам попробовать что-то новое.

Одни компании будут оплачивать ваши курсы, другие — нет. Мы призываем вас думать о том, что вы строите *свою* карьеру. Берите то, что вам нужно и вас интересует. Если вам придется потратить деньги, то

рассматривайте это как инвестицию. Приобретая новые навыки, вы станете более ценными либо для своей компании, либо для следующей (которая будет ценить вас более высоко).

Одни компании разрешат вам работать над проектами в другой команде на условиях полной или частичной занятости. Другие — нет, или же они будут держать вас на полной ставке в команде тестирования и на неполной — в другой. Второй случай наиболее распространен. Будьте готовы работать сверхурочно, чтобы иметь возможность заниматься временными проектами. Если вы не можете работать сверхурочно (возможно, семейные обязанности требуют, чтобы вы были дома), то можете найти время для прохождения курсов обучения со свободным графиком проведения и работы над проектами без работодателя (например, разработки открытого программного обеспечения) в ночное время. (Будьте внимательны и разделите ночное хобби и работу. Например, если вы напишете открытый код дома, а затем проведете тестирование или улучшение на работе, то ваш работодатель, вероятно, получит право собственности на него. Будьте осторожны и не создавайте ничего, что могло бы конкурировать с продуктами вашего работодателя.)

Расширяйте свою карьеру за пределы тестирования программного обеспечения

Многие из самых счастливых и успешных людей в области тестирования программного обеспечения переходят из одной сферы в другую.

Тратить свое время на поиск и критику чужих ошибок надоедает. Попробуйте что-нибудь другое. Развивайте навыки в другой области разработки программного обеспечения. Делайте свои ошибки. Затем, если захочется, вернитесь, чтобы снова заняться тестированием. Это даст вам возможность расширить перспективу, увеличить гарантию занятости, привлекательность на рынке труда, позволит быть более гибкими, лучше понимать проблемы и ограничения проектов, завоевать более высокий авторитет среди других членов команды разработчиков и, возможно, получить более высокую зарплату.

Люди, отвечающие за несколько функций, часто получают зарплату по другой, более высокой шкале, чем руководители, занимающиеся одной из функций. Во многих компаниях ценятся руководители, обладающие многофункциональностью.

Расширяйте свою карьеру за пределы компании

Это *ваша* карьера. Если завтра ваша компания разорится или уволит вас, то вы все равно останетесь собой и ваша карьера останется при вас. (Вам просто придется найти новую работу, и часто вы можете найти работу лучше той, которую только что покинули.) Существует большое сообщество тестировщиков ПО и еще большее сообщество разработчиков ПО. Мы пишем, встречаемся, спорим, многому учим друг друга, даем друг другу советы и находим друг другу новую работу.

Вы можете участвовать в жизни сообщества самыми разными способами. Посещайте конференции по тестированию ПО или разработке ПО. Присоединяйтесь к местному отделению Ассоциации по вычислительной технике, Американского общества по качеству, Общества человеческих факторов и эргономики или к любому другому профессиональному обществу, которое ценит качество программного обеспечения. Присоединяйтесь к списку рассылки software-test-discussion (www.testingfaqs.org/swtest-discuss.html) и участвуйте в нем, если вам есть что сказать. Если вы согласны с изложением принципов контекстно ориентированного тестирования, приведенным в конце этой книги, то зарегистрируйтесь (www.context-driven-software.com) и присоединитесь к нашему списку рассылки. Советы по созданию сетей можно найти в книге Агре (Agre, 2001).

Конференции предназначены для обсуждений

Когда вы посещаете конференцию по тестированию или разработке программного обеспечения, не просто сидите на заседаниях и слушайте докладчиков. Делайте это часто (или хотя бы частично). Но уделяйте много времени встречам с другими участниками конференции, чтобы *обсудить* представленные материалы или текущую ситуацию в данной области.

Если вы не знакомы с большинством участников конференции, то познакомьтесь с кем-нибудь. Когда вы идете на обед, сидите с незнакомыми людьми — слушайте и выявляйте интересных и знающих людей. Используйте и другие возможности для поиска интересных людей. Затем пригласите их на кофе или предложите угостить их завтраком. При встрече расспросите их о том, чем они занимаются, о заседаниях,

на которых они побывали. Что интересного они нашли? Со временем (для этого потребуется не одна конференция) у вас появится группа друзей, с которыми вы будете встречаться в основном на конференциях, поддерживать связь через Интернет, посещать заседания и, возможно, писать статьи или выступать с докладами. Так познакомились мы трое.

Не всех работников компания отправляет на конференции. Заранее сообщите своему руководителю, что хотите участвовать в конкретной конференции. Заблаговременное уведомление повысит ваши шансы на участие в ней. После того как вы посетите пару конференций, подайте заявку на участие в качестве докладчика. Если ее примут, то вероятность того, что ваша компания отправит вас, возрастает, к тому же (при условии хорошей работы) вы завоюете уважение и хорошую деловую репутацию в своей компании и за ее пределами.

Во многих других компаниях дела обстоят так же плохо, как и в вашей

Многие тестировщики приходят в ужас от ошибок в своих продуктах и неразберихи внутри компании. Это не такая уж необычная ситуация, даже в хороших компаниях. Тестировщики видят, что идет не так. Это не всегда весело.

По статистике, вы можете оказаться в 10 % худших компаний. Но, скорее всего, нет — в этих компаниях даже нет тестировщиков. Общение с тестировщиками в других компаниях может помочь вам увидеть перспективу.

Если вам не нравится ваша компания, то ищите другую работу

Как правило, целесообразно сохранить прежнее место работы; почти всегда легче найти хорошую работу, когда вы работаете. Не размещайте свое резюме в Интернете, если не хотите, чтобы его увидел ваш начальник. Общайтесь с ограниченным кругом рекрутеров и только с теми, кто согласен согласовывать с вами каждую отправку вашего резюме куда-либо.

Пройдя собеседование в других компаниях, вы можете согласиться на новую должность или решить, что компания, в которой вы работаете,

не так уж плоха. Решение сохранить имеющуюся работу может быть вполне адекватным и успешным результатом поиска работы.

Если ваша текущая работа настолько токсична, что отравляет ваше мнение о себе и то, как вы говорите о себе на собеседованиях, то, возможно, вам лучше уволиться, взять отпуск, а затем искать новую работу. Альтернативой увольнению из токсичной компании, особенно если вы не можете себе этого позволить, может стать решение о том, что вы будете выполнять свою работу добросовестно (честно, а не излишне тщательно), независимо от сроков и ограничений, или найдете другие способы повысить уважение к себе на работе. Если есть возможность, то учитесь на курсах по ночам. Высыпайтесь, если можете. Цель — привести себя в хорошее психическое состояние, чтобы наиболее эффективно представить свою кандидатуру следующему потенциальному работодателю.

Будьте готовы к тому, что вам придется поставить на кон свою работу (и проиграть)

Один из нас, Кейнер, всегда отслеживал ситуацию на рынке труда и обычно рассылал свое резюме всякий раз, когда работал в команде тестирования. Это не потому, что он хотел постоянно менять работу. Это был его способ защитить себя. Кейнер часто оказывался в ситуациях, требующих противоречивой позиции. Зная, что у него актуальное резюме, хорошие отношения с несколькими отличными рекрутерами, что он понимает, какие вакансии доступны, у кого и сколько платят, он мог быть уверенным в том, что если ему придется искать новую работу, то он сможет найти ее достаточно быстро. Зная *это*, он мог позволить себе пойти на риск, на который в противном случае не пошел бы.

Составьте список компаний, в которых хотели бы работать, и поддерживайте его в актуальном состоянии

Узнать о компаниях довольно просто, особенно с помощью Интернета, конференций и сетевого общения. Подумайте о том, где бы вы хотели работать. Познакомьтесь с людьми, которые там работают. Поговорите с ними.

Создайте портфолио

Очень удобно иметь комплект образцов кода, документов и других работ, демонстрирующих вашу компетентность. С его помощью можно продемонстрировать свою компетентность так, как это не могут сделать другие соискатели. Однако создание такого комплекта — непростая задача.

В большинстве компаний ваша работа — собственность компании и является конфиденциальной. Однако вы можете создавать некоторые документы или писать код, которые не являются особо секретными. Или же результаты вашей работы могут быть секретными сегодня, но не будут считаться таковыми пять лет спустя. Это кандидаты в ваше портфолио. Одни руководители с готовностью предоставят вам право на использование материалов, не являющихся конфиденциальными, но другие будут настроены более категорично. Получить разрешение на использование материалов, созданных вами в компании, вы можете в следующих ситуациях.

- Когда вас сократили, а ваш руководитель или представитель отдела кадров объясняет вам, какие выплаты полагаются при увольнении. Если у вас на руках есть конкретные образцы работы, то люди, с которыми вы обсуждаете увольнение, могут с удовольствием подписать документ, в котором говорится, что вы можете показывать эти образцы потенциальным работодателям или клиентам в целях получения работы. Получите письменное разрешение, а не только устное.
- Через полгода-год после увольнения, если ваш бывший начальник все еще работает в компании и вы хорошо ладите.
- Когда вы используете что-то в качестве примера на конференции. Если у вас есть разрешение на раскрытие секретной информации на конференции и вы это делаете, то для повторного раскрытия секрета вам не требуется дополнительного разрешения. Секрет больше не является секретом.

Если вы разрабатываете программное обеспечение или пишете статьи в свободное время, используя собственные ресурсы, не прибегая к ресурсам компании, и если материал, который вы разрабатываете, не связан напрямую с вашими задачами в компании, то вы, вероятно, являетесь владельцем того, что написали, и можете свободно использовать его. Вы можете обладать правами на свою работу при гораздо более широком диапазоне обстоятельств. Нюансы зависят от условий вашего договора с работодателем и регулирующих его законов.

Используйте свое резюме как инструмент продажи

Резюме позволяет вам отказываться от одних вакансий и претендовать на другие. Ваша цель — составить резюме, которое будет обращено к нужному субрынку.

- Если вас интересуют совершенно разные виды работ, то напишите несколько разных резюме. По-разному расставляйте в них акценты, чтобы подчеркнуть те навыки, интересы и факты биографии, которые представляют вас в наиболее выгодном свете. Следите за тем, чтобы описания вашей биографии и достижений были сопоставимы в разных вариантах резюме (всегда предполагайте, что работодатель в конечном итоге увидит все ваши резюме).
- Подготовьте хронологическое резюме. В таком резюме ваш опыт описывается в хронологическом порядке, при этом подчеркиваются ваши достижения на каждом месте работы. Некоторые рекрутеры (и работодатели) предпочитают функциональные резюме, в которых упор делается на опыт и навыки, а компания, в которой вы их получали, упоминается в качестве детали. В таких резюме нет ничего плохого, но некоторые люди используют их для того, чтобы преувеличить свой опыт или сокрыть периоды безработицы. В связи с этим некоторые менеджеры по подбору персонала (например, мы) настаивают на составлении исторического профиля кандидатов с указанием мест работы. Если вы не можете представить эту информацию в письменном виде, то мы вынуждены просить вас озвучить подробности по телефону или на очном собеседовании. Если же такая информация у вас есть, то во время телефонного собеседования, посвященного вашей рабочей истории, скажите, что можете отправить хронологическую версию вашего резюме по электронной почте. На одних менеджеров ваша подготовка произведет сильное впечатление. Другие менеджеры по подбору персонала игнорируют кандидатов, представляющих только функциональные резюме, если только навыки соискателей не соответствуют требованиям вакансии и их заявления не заслуживают доверия.
- Подготовьте функциональное резюме или резюме с ключевыми словами. В некоторых компаниях резюме сканируют или быстро просматривают люди, которые не понимают, на какую должность вы претендуете. Если вы хотите работать в одной из таких компаний, то вам необходимо вставить в резюме нужные слова или фразы. Для

этой цели хорошо подходят функциональные резюме. Как вариант, в хронологической версии можно добавить раздел «Ключевые слова: для компьютерного поиска», а затем в одном абзаце перечислить все применявшиеся языки программирования, важные навыки и т. д. Просто укажите названия, ничего не объясняя.

- Никогда не преувеличивайте свое образование, навыки или опыт. Ваш авторитет — один из главных активов. Если кто-то заподозрит, что вы преувеличили или солгали в своем резюме, то вас не возьмут на работу. Люди будут видеть ваше резюме и после того, как вас примут. Через два года после того, как вы устроились на работу, менеджер из другой команды (или ваш новый руководитель) может просмотреть ваше резюме и увидеть, что вы заявили о своей компетентности в чем-то, в чем явно не разбираетесь. Этот менеджер может не говорить об этом с вами, но наверняка поговорит с другими менеджерами. Это не поможет вашей карьере.

Получите рекомендацию сотрудника компании

Многие компании предоставляют своим сотрудникам бонусы за рекомендации кандидатов, которых берут на работу. Воспользуйтесь этим. Пусть вас порекомендует действующий сотрудник компании, в которую вы устраиваетесь. Таким образом у вас будет возможность получить более достоверную информацию о компании и, вероятно, ценную обратную связь в процессе собеседования и переговоров. Нам это помогало.

Изучите данные о зарплатах

Получить реалистичные данные относительно размера заработной платы можно с помощью таких баз данных, как www.salary.com. Знайте эту информацию и идите на собеседование подготовленными.

Если вы отвечаете на объявление, то адаптируйте свой ответ

Если в объявлении указано несколько требований, то ответьте на них. Укажите те из них, которым вы соответствуете, и те, по которым у вас нет опыта, но вы очень хотите научиться.

Пользуйтесь возможностью пройти собеседование

Когда вы (или рекрутеры) рассылаете свое резюме, вам могут предложить пройти собеседование на вакансию, на которую вы вряд ли согласитесь. Возможно, эта работа недостаточно хорошо оплачивается, или расположена там, где вы не хотели бы жить, или же не впечатляет вас по каким-либо другим причинам. Независимо от этого, подумайте о том, чтобы согласиться на собеседование, и ведите себя на нем так, будто действительно хотели бы получить эту работу. Собеседование — это возможность отработать навыки общения с работодателем в непринужденной, но реалистичной обстановке.

Вы сможете попрактиковаться в ответах на вопросы собеседования и услышать новые вопросы, к которым не были готовы заранее. Если вы новичок или не имеете опыта в поиске работы, то будете нервничать во время первых собеседований. Поучаствовав в нескольких, вы будете лучше представлять себе, чего ожидать, и сможете более спокойно относиться к этому процессу. Собеседования по поводу вакансий, на которые вы не собираетесь соглашаться, даже если они будут вам предложены, все равно помогут вам получить опыт прохождения собеседований и обрести уверенность в себе.

Возможно, вы обнаружите, что данная вакансия вас все-таки устраивает или в компании есть другие вакансии, которые вы предпочтете.

Узнавайте больше о компаниях, подавая заявку на работу в них

Если вы безработный и рассылаете много резюме, то у вас нет времени изучать каждую компанию. Так что можете этого не делать. Но, увидев особенно интересную вакансию, вы можете провести огромное количество исследований в Интернете бесплатно и еще больше — заплатив несколько долларов. Зайдите на сайт компании, почитайте их материалы, загрузите все демонстрационные программы, которые они предоставляют, поищите информацию об используемых ими средствах программирования. Используйте поисковые системы. Если это открытое акционерное общество, то зайдите на сайты, предоставляющие информацию для инвесторов, и задайте вопросы. Чем больше вы знаете, тем более уникальным вы сможете сделать свое сопроводительное письмо.

Как только вы узнаете, что компания (которая вам интересна) проявляет к вам интерес, вам следует изучить ее. Чем больше вы будете знать, тем эффективнее будете вести с ними переговоры.

Если кто-то приглашает вас на собеседование по телефону, то попросите предоставить:

- литературу с описанием продукции или компании;
- демонстрационные копии издаваемых компанией программ или спросите, какие инструменты они используют для разработки и тестирования своих программ и какие типы приложений создают;
- данные о том, где можно найти другую информацию о компании.

Если они пришлют вам какие-либо материалы, то обязательно прочитайте их перед личным собеседованием.

Люди, проводящие собеседование, желают знать, хотите ли вы работать в их компании и почему. Многие менеджеры по подбору персонала отдают предпочтение стремящимся работать энтузиастам, обладающим достаточной квалификацией, а не соискателям с высокой квалификацией, но относящимся к предстоящей работе без особого энтузиазма. Воодушевленный сотрудник сможет освоить свою работу, и с ним будет интереснее работать. (Возможно, он к тому же будет работать усерднее.) Просто сказать: «О-о-о, о-о-о, я в восторге!» — менее убедительно, чем показать, что вы читали о компании и серьезно интересуетесь тем, что она делает. Ваши действия (в данном случае исследование) говорят лучше слов.

Задавайте вопросы на собеседованиях

Неплохо задавать вопросы и во время собеседования (проводимого по телефону или при личной встрече), хотя, вероятно, не стоит задавать слишком много вопросов одному человеку, а для разных людей следует подбирать подходящие вопросы. Вот несколько примеров вопросов, на которые часто дают интересные ответы.

- Какие виды продуктов и услуг предоставляет ваша компания?
- Могу ли я увидеть демонстрацию основного продукта?
- Что особенного в ваших продуктах и услугах? Каковы основные сильные и слабые стороны?

- Как вы разрабатывали основной продукт? В чем заключались ключевые компромиссы при разработке?
- Кто ваши клиенты?
- Кто ваши конкуренты?
- Как вы узнаёте о своих клиентах?
- Как вы узнаёте, удовлетворены ли клиенты качеством продукта, его дизайном и возможностями?
- Опишите вашу организационную структуру (где в ней находитесь вы и где в ней находился бы я).
- Каково это — работать здесь?
- Чем вы занимаетесь? Какие виды продуктов и услуг предоставляете? (Вопрос обращен человеку, с которым вы разговариваете на собеседовании.)
- Можно ли посмотреть примеры?
- Какое место вы занимаете в процессе разработки продукта?
- Что вам нравится в вашей работе?
- Что бы вы хотели изменить?
- Как вы находите время для своей семьи?
- Насколько вы контролируете свою работу?
- Кто разрабатывает тесты, которые вы проводите? Кто проводит тесты, которые вы разрабатываете?
- Расскажите о процессе разработки тестов.
- Могу ли я ознакомиться с тест-планами и тестовыми случаями?
- Как вы относитесь к своей зарплате, начальству, коллегам?
- Какие курсы или конференции вы посещали в прошлом году?
- Какое еще обучение вы прошли?
- Как вы учитесь новому?
- Опишите три ключевых момента, которым вы научились в прошлом году.

Во время личных собеседований отслеживайте следующие моменты.

- Устали ли те, кто проводит собеседование?
- Положительно ли отзываются о своей работе эти люди и другие сотрудники?
- Ощущается ли моральный дух и чувство сплоченности коллектива?

- Как обстоят дела с мебелью? Сколько средств вкладывается в комфорт и продуктивность инженерно-технического персонала? Как выглядит их мебель в сравнении с мебелью менеджеров или руководителей?
- Сколько места, оборудования и света получают рабочие тестировщики и сколько — сотрудники более высокого ранга?
- Ищите несоответствия между заявлениями об условиях труда и реальными условиями. Например, некоторые компании в своих объявлениях говорят о 4-дневной рабочей неделе с 10-часовым рабочим днем, но при этом ожидают, что в остальные три дня (те, которые у вас должны быть выходными) вы будете много работать. Подобные несоответствия порождают недовольство и неприятную атмосферу. Они также могут рассматриваться как признаки этических проблем у руководства компании: если оно не говорит прямо о своих основных ожиданиях, то о чем еще умалчивает?

Ведите переговоры о вашей позиции

В данной книге мы лишь поверхностно коснемся этой темы. Ведение переговоров — это навык. Вы должны практиковать его. Делайте это, общаясь с друзьями и потенциальными работодателями. (Вы потеряете несколько потенциальных мест работы, но будете учиться на своих ошибках.)

Чем лучше вы информированы во время переговоров, тем эффективнее ведете их. Ключевой является информация о том:

- чего хотите вы;
- чего хотят представители компании;
- что они думают о себе;
- как они будут представлять себя в роли потенциальных работодателей;
- каковы ваши альтернативы в целом; какова ваша лучшая альтернатива этой работе.

Наличие альтернатив очень важно в переговорах. Если вы работаете по найму, то самая простая альтернатива — сохранить свою нынешнюю работу и ждать, пока не появится более подходящая. Это ключевая причина, почему людям, имеющим работу, легче найти хорошую работу, чем тем, кто ее не имеет. Безработные редко имеют столь же

надежную и очевидную альтернативу, как те, у кого есть работа. Они часто находятся в отчаянии, и, даже если это не так, их воспринимают как людей, которым в ближайшее время нужна работа, иначе у них закончатся сбережения (и они впадут в отчаяние). В результате компании часто считают, что могут предложить меньше денег и дать меньше других обещаний человеку, не имеющему работы. Если вы безработный, то не замедляйте поиск работы, когда связались с интересной вам компанией. Ускорьте его. Найдите другую привлекательную компанию. Независимо от того, сообщите ли вы каждой компании о другой (будьте осторожны, называя одну компанию в другой, некоторые считают это нарушением конфиденциальности), вы будете вести переговоры иначе, поскольку будете знать, что у вас есть альтернативные варианты. Уверенность, которую дает это знание, повлияет на то, как вы себя преподнесете.

В ходе переговоров необходимо обсуждать следующие моменты.

- В самом начале собеседования расскажите о том, чего вы хотите (неденежные вопросы) и что вас привлекает. Если вы заинтересуете представителей компании, то они проведут собеседование (и остальные переговоры) таким образом, чтобы убедить вас присоединиться к компании.
- Откажитесь предоставить информацию о предыдущей зарплате. Вежливо отклоняйте вопросы о ваших ожиданиях относительно оплаты до тех пор, пока работодатель не заинтересуется вами. Часто бывает полезно объяснить, что ваша предыдущая зарплата не имеет отношения к данной должности, поскольку она была получена за другую работу. Вы хотите получить справедливую цену за *предстоящую* работу. Затем спросите, какова шкала оплаты труда на этой должности. Одни сотрудники отдела кадров ответят вам. Другие откажутся, и тогда вы сможете сказать: «Давайте поговорим о деньгах позже, после того как обсудим обязанности и возможности».
- Будьте полны энтузиазма, но не переусердствуйте.
- Расскажите об их продукте, говоря на их языке.
- Найдите способы, которыми вы можете им помочь, и опишите их. Вносите предложения, показывайте примеры своего мышления.
- Дайте им понять, что вы хотите получить эту работу. (В некоторых культурах уместнее сказать, что работа очень интересная и кажется очень желанной, но важно не говорить, что вы хотите ее получить, пока не будете готовы принять их предложение.)

- Если вы не обладаете достаточной квалификацией для этой работы, но хотите ее получить, то откровенно расскажите о ней как о возможности для роста. Это вызов, то, чего вы не делали раньше, то, для чего вам придется развить новые навыки и знания, но то, что вы хотите сделать. Пусть ваши собеседники увидят, как они могут сделать вас счастливыми, как эта работа соответствует вашим интересам и целям и как вы можете вырасти, занимаясь ею.
- Следите за стилем ведения переговоров со стороны сотрудников потенциального работодателя: не является ли он угрожающим.

Помните, что во время собеседования и переговоров менеджеры компании ведут себя наилучшим образом. Они не станут более учтивыми, добрыми, честными или внимательными после того, как вы начнете работать в компании.

Хорошие книги по переговорам написали Чепмен (Chapman, 1996), Фишер и Эртель (Fisher, Ertel, 1995), Фишер и др. (Fisher et al., 1991), Фрейнд (Freund, 1992), Миллер (Miller, 1998), О'Мэлли (O'Malley, 1998) и Таррант (Tarrant, 1997). Особенно удачные (на наш взгляд) рекомендации по ведению переговоров привел Фрейнд.

Будьте осторожны, общаясь с сотрудниками отдела кадров

Специалисты по работе с персоналом не принимают решений о найме, хотя часто решают, с какими резюме могут ознакомиться менеджеры по найму. Функциональные менеджеры (например, менеджер по тестированию или разработке) решают, кто в конечном итоге будет принят на работу. Ведите переговоры с лицом, принимающим решение, а не с отделом кадров.

Изучайте язык Perl

Perl — удобный скриптовый язык, и многие программисты в вашей компании тоже его знают. (Так что вы можете попросить у них совета, если чего-то не поймете.) На Perl можно написать множество служебных программ. Например, можно анализировать файлы журналов, передавать данные на устройства или разрабатывать автоматизированные тесты и передавать им данные. Чем больше вы работаете с этим языком, тем больше новых идей о том, как еще его можно использовать, у вас появится. (Примечание: если Perl не выступает в роли скриптового

языка в вашей компании, то используйте тот, который является таковым, или тот, который вам больше нравится. В качестве альтернативы часто упоминаются Python и Ruby. Если вы не знаете, с чего начать, и у вас нет причин начинать с другого языка, то мы предлагаем начать с Perl.) Более подробную информацию о скриптовых языках можно найти на сайте www.softpanorama.org/Scripting/index.shtml.

Изучайте язык Java или C++

Если у вас есть возможность активно применять полученные знания, например писать собственные инструменты тестирования, изучите основной язык разработки в вашей компании и практикуйтесь на нем. Чем лучше вы будете владеть языком, тем больше возможностей у вас будет в компании и при выборе следующей работы.

Скачайте демонстрационные копии инструментов тестирования и опробуйте их в деле

Возможно, вам придется делать это дома, в свободное от работы время. Мы предлагаем вам ознакомиться с инструментами, которые ваша компания еще не использует (пока). Возможно, ваша компания не захочет платить вам за эти исследования или поручит вам выполнять эту работу вместо основных проектов.

После того как вы узнаете о существующем инструменте и о том, насколько хорошо он работает, вы сможете убедить свою компанию приобрести его и начать использовать. Если он хорош в деле, то люди оценят вашу рекомендацию. После нескольких хороших рекомендаций люди могут начать смотреть на вас как на исследователя групповых инструментов.

Независимо от того, ценит ли ваш нынешний работодатель то, что вы постоянно обновляете инструменты тестирования, эти знания помогут вам при собеседовании с потенциальными работодателями.

Совершенствуйте навыки письма

Бóльшая часть вашей работы связана с составлением отчетов и служебных записок, призванных убедить других людей в необходимости

совершения тех или иных действий. Чем лучше вы пишете, тем эффективнее ваши отчеты. Пройдите курсы по техническому письму и искусству литературного убеждения и найдите другие способы практиковаться и совершенствоваться в написании текстов.

Совершенствуйте навыки публичных выступлений

Курсы ораторского искусства можно пройти во многих университетах и муниципальных колледжах. Кроме того, можно вступить в организацию под названием *Toastmasters* (www.toastmasters.org). Она обеспечивает безопасную и благоприятную среду для обучения и отработки навыков публичных выступлений. Многие клубы Toastmasters — хорошие места для общения; вы можете удивиться, узнав, кто еще находится в зале (руководители, менеджеры, другие инженеры и т. д.).

Подумайте о получении сертификата

Кейнер имеет сертификат Американского общества контроля качества (American Society for Quality). Сертификацию в области тестирования и управления качеством ПО можно пройти в Институте обеспечения качества (Quality Assurance Institute). Аналогичную возможность дает Британское компьютерное общество (British Computer Society). Пройти сертификацию можно и в других группах.

Стоит ли оно того?

У нас сложилось впечатление, что сдать сертификационные экзамены не так уж сложно. Вопросы часто достаточно предсказуемы. Если вы пройдете курс подготовки к экзамену, то, скорее всего, успешно сдадите его. Некоторые группы время от времени публикуют в Сети примеры вопросов. По нашим наблюдениям, экзамены обычно состоят из нескольких вариантов ответов, а иногда и из очень легких вопросов. Часто экзаменаторы проверяют знание лексики, относительно простых понятий и базовых навыков. Наличие сертификата не означает, что человек является экспертом.

Многие из тех, кто получает сертификат, делают это после прохождения ряда курсов, а не только обзорного. Они используют подготовку к сертификации как возможность расширить свое образование. По нашему

опыту, эти люди ценят свою сертификацию и считают, что получили много пользы от этого процесса.

Сертификация может быть полезна при составлении резюме. Она говорит потенциальному работодателю о том, что вы относитесь к данной области достаточно серьезно, чтобы пройти курс повышения квалификации и сдать сертификационный экзамен. Это выгодно отличает вас от большинства других людей, претендующих на любую должность тестировщика. Однако если вы сертифицированы (но не являетесь программистом), а кто-то другой — нет, но умеет программировать, то не удивляйтесь, если работу по автоматизации тестирования получите не вы. Ваша сертификация может помогать вам только до определенного момента.

В целом мы считаем, что усилия, вложенные в тестирование ПО и сертификацию его качества, пошли профессии на пользу. Они побудили многих людей изучить классические книги и работы в этой области, освоить новые техники и повысить свою квалификацию относительно решения вопросов, связанных с качеством ПО.

Мы не одобряем систему знаний или взгляды сертифицирующих организаций. У нас с ними есть разногласия, но эти организации имеют право на свои взгляды и на определение квалификации людей. Существенное различие между сертификацией (против которой мы не возражаем) и лицензированием (против которого мы возражаем) заключается в том, что сертификация более скромна. Вы можете практиковать свое ремесло независимо от того, прошли ли вы сертификацию. Вы можете выбрать ту сертификацию, которая кажется вам наиболее полезной или здравой. Это средство обучения, а не средство регулирования (см. урок 273).

Следует отметить одну проблему. Если вы получили сертификат и являетесь независимым консультантом или подрядчиком, то будьте осторожны в том, как рекламируете свои услуги. Если вы представите себя как сертифицированного специалиста в области контроля качества (или по другой специальности), то вас могут посчитать тем, кто обладает определенной квалификацией и предоставит свои услуги на том уровне, который следует ожидать от человека, обладающего таким уровнем квалификации. Если вы некачественно выполните свою работу, то вашим клиентам будет легче предъявить вам иск за халатность или недобросовестное исполнение своих обязанностей (Kaner, 1996a). Поэтому, даже имея сертификат, вы, возможно, не захотите рекламировать себя таким образом.

Если вы смогли получить черный пояс всего за две недели, лучше избегайте драк

Люди могут зарабатывать на сертификационном бизнесе — на обзорных курсах, учебных пособиях, а также на самом экзамене. Многие люди заинтересованы в сертификации, которая имеет мало общего с вашей успешной работой в качестве инженера.

Значение некоторых сертификатов кажется нам сильно преувеличенным. Мы с подозрением относимся к заявлениям программ о том, что они могут сделать из вас мастера, эксперта или обладателя черного пояса за пару недель.

В боевых искусствах получение черного пояса занимает много времени. Первым этапом часто является получение желтого пояса, который инструктор дает вам, когда считает, что вы более опасны для окружающих, чем для себя. Даже на это обычно уходит гораздо больше времени, чем две недели.

Может быть, дисциплина настолько проста, что ее можно освоить за две недели; если да, то, конечно, получайте свой черный пояс. Но не ввязывайтесь в драки.

Предупреждение о попытках лицензирования инженеров-программистов

Должны ли инженеры-программисты (в том числе тестировщики) получать лицензию? Другие виды инженеров лицензируются, может быть, и мы должны?

Лицензия на инженерную деятельность выдается государством. В США лицензии выдаются правительствами штатов, поэтому если вы работаете в двух штатах, то получаете две лицензии. Лицензированные специалисты должны соблюдать этический кодекс и могут быть привлечены к суду за недобросовестную работу.

В настоящее время предпринимаются усилия по введению требования о лицензировании инженеров-программистов. Эти усилия увенчались успехом в Техасе, Британской Колумбии и Онтарио. Отчет о текущем состоянии см. на сайте Construx, посвященном профессиональной деятельности инженеров-программистов, www.construx.com/profession/home.htm. Мид (Mead, 2001) приводит продуманные аргументы в пользу лицензирования.

Одна из организаций, Координационный комитет по программной инженерии (Software Engineering Coordinating Committee, SWECC), занимается разработкой Свода знаний по программной инженерии (Software Engineering Body of Knowledge, SWEBOK). SWECC был совместным комитетом Компьютерного общества Института инженеров по электротехнике и электронике (Institute for Electrical and Electronics Engineers Computer Society, IEEECS) и Ассоциации вычислительной техники (Association for Computing Machinery, ACM). Целью проекта SWEBOK было достижение консенсуса по поводу формулировки знаний, которыми должны обладать все инженеры-программисты.

ACM не поддерживает лицензирование инженеров-программистов и поэтому в июне 2000 года отказалась от участия в работе над SWEBOK и от участия в SWECC.

> «Позиция ACM заключается в том, что состояние знаний и практики в области программной инженерии слишком незрелое, чтобы оправдать лицензирование. Более того, Совет [ACM] считает, что лицензирование будет неэффективным в деле обеспечения гарантий качества и надежности программного обеспечения… Совет пришел к выводу, что система профессионального лицензирования, изначально разработанная для инженеров-строителей, не соответствует производственной практике в программной инженерии. Такая практика лицензирования давала бы ложные гарантии компетентности, даже если бы свод знаний был полным, и не позволила бы многим из наиболее квалифицированных инженеров-программистов получить лицензию… В связи с тем что она стала тесно ассоциироваться с лицензированием инженеров-программистов в рамках модели профессионального инженера, Совет ACM принял решение о выходе из SWECC» (ACM, 2000).

SWEBOK — важный аспект движения за лицензирование. SWEBOK должен представлять для вас интерес, поскольку тестирование ПО является одним из направлений SWEBOK, а качество ПО — другим. Вы можете скачать его с сайта www.swebok.org.

Чтобы получить лицензию инженера-программиста, необходимо сдать экзамен. Он должен быть основан на знаниях и практиках, широко распространенных в данной области. После получения лицензии вам может быть предъявлен иск за недобросовестное исполнение своих обязанностей. Специалист совершает халатность, если причиняет вред своим клиентам (например, они получают физические травмы, повреждается их имущество или они теряют деньги) из-за того, что он не

использовал свои навыки или не применил знания, которыми должен обладать как грамотный представитель своей профессии. Если SWEBOK будет принят в качестве Свода знаний по программной инженерии, то законодатели, создатели экзаменов, судьи, адвокаты, присяжные и газетные репортеры будут обращаться к нему, когда захотят разобраться в стандартах, определяющих, что должны знать и делать инженеры-программисты.

Мы подвергли критике утверждения, содержащиеся в SWEBOK, в предисловии и в главе 6. И мы не одиноки. Более широкая негативная оценка содержится в докладе Ноткина и др. (Notkin et al., 2000), представленном Совету ACM.

В предисловии к Своду знаний по программной инженерии (IEEE Computer Society, пробная версия 0.95) говорится следующее:

> «Цель настоящего Руководства — дать утвержденную на основе консенсуса характеристику границ дисциплины программной инженерии и обеспечить тематический доступ к Своду знаний, поддерживающих эту дисциплину… Акцент на инженерной практике приводит к тому, что Руководство тесно связано с нормативной литературой. Бóльшая часть литературы по информатике, информационным технологиям и программной инженерии содержит сведения, полезные для инженеров-программистов, но относительно небольшая часть носит нормативный характер. Нормативный документ скорее предписывает, что должен делать инженер в той или иной ситуации, чем предоставляет информацию, которая может быть полезной. Нормативная литература подтверждается консенсусом, сформированным среди специалистов-практиков, и делает акцент на стандартах и сопутствующих документах. Проект SWEBOK с самого начала задумывался как имеющий тесную связь с нормативной литературой по программной инженерии… В итоге мы надеемся, что стандарты практики программной инженерии будут содержать принципы, соответствующие Руководству SWEBOK».

В предисловии к SWEBOK дается определение консенсуса:

> «Под консенсусом мы подразумеваем, что единственный практический способ легитимизации заявления такого рода — широкое участие и согласие всех значимых секторов соответствующего сообщества».

Для нас немыслимо, чтобы документ, от которого отказалось главное сообщество профессионалов в области компьютерных технологий в США, мог считаться консенсусным. На наш взгляд, многие нормативные

положения SWEBOK не описывают, как поступил бы грамотный инженер в той или иной ситуации.

Мы считаем, что опасно относиться к SWEBOK так, будто это консенсусный документ. Если считать, что SWEBOK описывает стандарт профессиональной деятельности и знаний, то что будет с инженером-программистом, который участвовал в провальном проекте и не следовал подходам, рекомендованным в SWEBOK? Такое нарушение может быть истолковано как недобросовестное исполнение своих обязанностей, даже если инженер применял методы, которые на самом деле были более подходящими в данных обстоятельствах.

Кто решает, что неприменение SWEBOK в конкретном случае является халатностью? Не другие инженеры. Решение принимают судьи, присяжные, адвокаты и страховые компании — люди, не имеющие или почти не имеющие инженерного опыта. Их решение будет оказывать огромное влияние в данной области.

Что мы должны ожидать в результате, если дадим неинженерам свод знаний, предписывающий нормы, которые в настоящее время не соблюдаются, которые (по нашему опыту работы со многими крупными и мелкими компаниями по разработке ПО) не рассматриваются как авторитетные, на них даже не ссылаются и их не читают большинство практиков, и которые, как представляется, зачастую подкреплены не более эмпирическими доказательствами их достоинств, чем подсчет голосов той ничтожной части специалистов, которая участвовала в создании конкретного стандарта? Мы ожидаем, что вердикты по делам о недобросовестном исполнении обязанностей будут случайным образом связаны с качеством принятия решений инженерами, о которых идет речь.

Суды в США постоянно отказываются принимать к рассмотрению иски о недобросовестном исполнении обязанностей в области компьютерных технологий, поскольку разработка ПО и программная инженерия не являются профессиями; а если человек не является представителем профессии, то на него нельзя подать в суд за профессиональную халатность (недобросовестное исполнение своих обязанностей) (Kaner, 1996a). Мы считаем, что до тех пор, пока в этой области не будет достигнуто широкое и подлинное согласие в отношении навыков и процессов принятия решений, которые приводят к правильному выбору практик, было бы опрометчиво объявлять эту область профессией и бросать ее членов на произвол судьбы в рамках правовой системы, регулирующей недобросовестное исполнение обязанностей.

Кстати, страхование профессиональной халатности для инженеров-программистов сегодня не очень дорого, поскольку вероятность того, что вам предъявят иск за нее, очень мала. В тех областях, где судебные разбирательства по поводу недобросовестного исполнения обязанностей носят серьезный характер, страховые премии исчисляются тысячами, а иногда и десятками тысяч долларов в год. Многие штаты требуют, чтобы лицензированные специалисты страховали профессиональную ответственность, если занимаются своей деятельностью в данном штате. Сколько вы хотите тратить на страховку, чтобы иметь право работать в качестве независимого подрядчика или консультанта?

Наши заявления о халатности иногда неверно истолковываются как стремление защитить инженеров, некачественно выполняющих работу. Мы хотели бы прояснить этот вопрос. Мы неравнодушны к качеству программного обеспечения. Мы решительно поддерживаем законы, предусматривающие ответственность разработчиков, в том числе компаний-разработчиков, за плохую работу. Мы все трое активно выступали против Единообразного закона о сделках с компьютерной информацией, поскольку он защищает разработчиков и продавцов ПО от ответственности за выпуск некачественных продуктов. Книга Кейнера *Bad Software*, посвященная защите прав потребителей программного обеспечения, была оценена Ральфом Нейдером как «руководство по защите прав потребителей в информационную эпоху» (Kaner, Pels, 1998, задняя обложка).

Мы выступаем против лицензирования инженеров-программистов не потому, что считаем иски за халатность плохой идеей в принципе. При правильных обстоятельствах такие иски служат прекрасным инструментом контроля компетентности в той или иной профессии. К сожалению, сегодня эти обстоятельства неприменимы к программному обеспечению, и ничто из того, что мы можем пожелать или сказать, не избавит нас от этой проблемы.

Мы выступаем за ужесточение стандартов для программного обеспечения. В качестве очень простого примера можно привести требование о раскрытии информации.

Поставщик программного обеспечения (разработчик, издатель или посредник) должен быть обязан сообщать покупателю ПО о каждом дефекте продукта, который известен поставщику на момент поставки продукта клиенту. Поставщик должен нести ответственность перед клиентом за любые убытки, вызванные дефектами, о которых было известно, но не сообщено клиенту таким образом, чтобы эту информацию мог понять обычный покупатель данного продукта.

Мы считаем, что это скромное требование окажет существенное влияние на качество продукции, представленной на рынке.

Мы выступаем за правовые стандарты, которые связаны с ущербом и потерями, вызванными дефектами продукта и ошибочными (ложными) заявлениями продавца. Если ввести ответственность за плохие результаты, то люди будут исправлять свои процессы, чтобы добиться лучших результатов. Несколько лет работы в *таком* режиме — и мы вполне можем достичь всеобщего признания набора практик разработки. В это время будет уместно пересмотреть вопрос о профессионализации.

Глава 11

Разработка стратегии тестирования

Тест-план — это набор идей, которыми вы руководствуетесь в процессе тестирования. Мы используем термин *«стратегия тестирования»* для обозначения набора идей, которыми руководствуются при разработке тестов, на протяжении всего проекта. Стратегия тестирования — важная часть хорошего тест-плана. Она служит связующим звеном между тестированием и поставленной целью. В учебниках много говорится о логистике и результатах тестирования, но не так много о стратегии, поэтому в данной главе мы сосредоточимся именно на ней.

Три основных вопроса, которые следует задать о стратегии тестирования: «зачем?», «кому это важно?» и «сколько?»

В конечном счете есть только одна причина, по которой вы проводите тестирование: что-то важное может пойти не так. Процесс тестирования существует для того, чтобы выявлять и исследовать риски, которые могут привести к отказу продукта, и сообщать о них. Именно поэтому необходимо постоянно задавать три вопроса о стратегии тестирования.

- *«Зачем?»* Тестирование — это дорого. Не включайте в свою стратегию мероприятия, если они не направлены на устранение рисков, которые достаточно важны, чтобы тратить время на их тестирование.
- *«Кому это важно?»* Причины для тестирования — это не законы природы; они коренятся в чувствах и ценностях людей, которые

имеют значение. Не включайте в свою стратегию мероприятия, если они не отвечают чьим-то интересам.

- *«Сколько?»* Некоторые стратегии гораздо легче обозначить словесно, чем реализовать. «Мы протестируем все комбинации характеристик принтера» — это одно короткое предложение, которое запускает тысячу тестов (или сто тысяч). Сколько из них вы действительно собираетесь сделать?

Существует множество возможных стратегий тестирования

Стратегия тестирования — это набор вариантов. В вашем распоряжении может быть много вариантов. Опишем несколько альтернативных стратегий.

- Проведя краткий внутренний анализ для выявления действительно серьезных проблем, мы передаем продукт на тестирование дружественным пользователям. Они введут продукт в эксплуатацию и расскажут нам о тех изменениях, которые им бы хотелось видеть.
- Мы определим сценарии использования в виде последовательностей взаимодействия пользователей с продуктом, которые в целом представляют все способы применения продукта обычными людьми. В дополнение к этому мы проведем стресс-тестирование и тестирование аномального использования (недопустимые данные и условия ошибок). Наш главный приоритет — поиск фундаментальных отклонений от заданного поведения, но мы также озабочены тем, как эта программа может нарушить ожидания пользователей. Надежность тоже вызывает озабоченность, но мы еще не решили, как ее лучше оценить.
- Мы будем проводить параллельное исследовательское тестирование, а также разработку и выполнение автоматизированных регрессионных тестов. Исследовательское тестирование будет проводиться с учетом рисков и распределяться по областям покрытия по мере необходимости. Мы будем пересматривать распределение каждую неделю. Автоматизированное регрессионное тестирование будет сосредоточено на проверке основных функций (функциональное тестирование), чтобы обеспечить систему раннего предупреждения о серьезных функциональных отказах. Мы также будем следить за возможностью проведения крупномасштабного выборочного тестирования.

Каждое из вышеперечисленных действий — стратегия. Обратите внимание, что все они разные. Каждая из стратегий имеет основную идею. Каждая кратко рассказывает, как мы собираемся проводить тестирование. Кроме того, хорошая стратегия убедительно объясняет и оправдывает необходимость проведения тестирования. Как существует множество историй, которые можно рассказать, так существует и множество различных стратегий тестирования.

Отметим, что каждая из стратегий является достаточно общей. В реальном проекте мы бы использовали наши конкретные знания о продукте для разработки целевых стратегий тестирования. Тем не менее даже эти общие абзацы показывают, что стратегия — это нечто большее, чем перечень техник, и в то же время меньшее, чем полноценный тест-план.

Реальный тест-план — это набор идей, которые направляют процесс тестирования

Ваш тест-план — это *идеи, которыми вы руководствуетесь в своей работе*. То, что у вас есть такие идеи, очень важно. А вот документированы ли они и каким образом — совершенно отдельный вопрос.

Всякий раз, когда вы слышите аргумент о важности создания документации для тех процессов, которые вы собираетесь осуществлять, помните о недостающем прилагательном между словами «создать» и «документация»: должна ли это быть хорошая или плохая документация? Большинство людей согласятся с тем, что важно *не* создавать *плохую* документацию. Но проблема заключается в следующем: хорошую документацию сложно написать. Ее дорого и трудно поддерживать.

Существует большое количество литературы по тестированию, в которой, по сути, говорится, что «без письменного тест-плана невозможно провести тестирование». По нашему опыту, основным положительным эффектом этого совета является повышение гарантий занятости производителей бумаги и тонеров. Существует слишком много плохо написанных планов. И мы видели много хороших тестирований, проведенных без соблюдения написанных планов. Пришло время дать вам лучший совет.

Не путайте содержимое плана со средствами, с помощью которых вы передаете информацию и управляете планом. У вас есть множество вариантов, помимо официального письменного трактата: устный план,

план на доске, план на одной странице, серия электронных писем, набор набросков или список вопросов. Делайте то, что соответствует цели.

Разрабатывайте тест-план в соответствии с контекстом

Один из способов визуализации планирования тестирования показан на рис. 11.1. Это контекстная модель удовлетворения. Пять овалов на концах звезды обозначают то, что вам дано: ресурсы и ограничения. Центр звезды представляет ваш выбор. Цель планирования — сделать выбор в отношении процесса тестирования, который позволит вам провести тестирование в рамках ограничений проектной среды, используя ресурсы, и таким образом достичь поставленной цели.

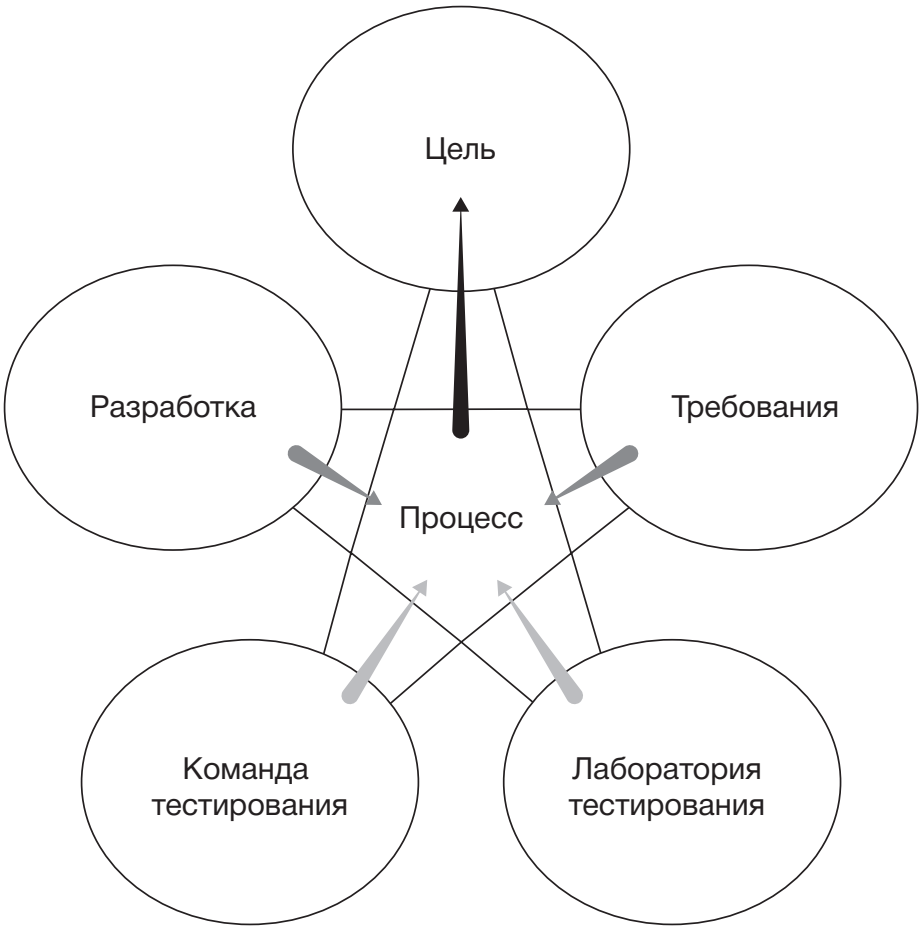

Рис. 11.1. Контекстная модель удовлетворения

Рассмотрим то, что нам дано.

- *Разработка.* Система, производящая продукт, который вы будете тестировать. Как вы получаете продукт? Насколько он поддается тестированию?
- *Требования.* Критерии успешного продукта. Каковы риски продукта? Чье мнение о качестве имеет значение?
- *Команда тестировщиков.* Люди, готовые тестировать продукт. Есть ли у вас подходящий персонал? Насколько хорошо они владеют технологией?
- *Лаборатория тестирования.* Системы, инструменты и материалы, которые позволяют вам выполнять свою работу. Имеется ли у вас необходимое оборудование? Исправна ли ваша система отслеживания багов?
- *Цель.* Проблемы, которые вы должны решить, чтобы ваши клиенты считали вас успешными. Быстро находить важные баги? Производить точную оценку качества?

Возможно, вам удастся договориться о более выгодных условиях. Вы можете нанять персонал или убедить программистов сделать продукт, более пригодный к тестированию. Однако не стоит рассчитывать на то, что вы сможете контролировать все это. Контроль заключается в том, как вы реагируете на ситуацию: какими будут ваша стратегия, логистика и результаты работы?

Используйте тест-план, чтобы обозначить выбор стратегии, логистики и результатов работы

Хороший план, независимо от того, документирован он или нет, содержит набор решений, касающихся процесса тестирования. Три основных категории описывают выбор, который вы должны сделать.

- *Стратегия.* Как вы будете покрывать продукт, чтобы быстро находить важные проблемы? Что конкретно вы будете тестировать? Какие техники будете использовать для создания тестов? Как вы будете распознавать баги при их возникновении? Стратегия тестирования определяет взаимосвязь между проектом и целью тестирования.
- *Логистика.* Как вы будете использовать ресурсы для реализации стратегии? Кто будет проводить тестирование? Когда эти люди будут это делать? Что вам нужно для успешного проведения тестирования?

- *Результаты работы.* Как ваша работа будет представлена клиентам? Как вы будете отслеживать баги? Какую тестовую документацию будете создавать? Какие отчеты будете составлять?

Если вы не делаете этот выбор ясно в плане тестирования, то делаете его скрытым образом с помощью других средств. У вас нет возможности не выбирать, если только вы просто не откажетесь от тестирования.

Не позволяйте логистике и результатам работы затмить стратегию

Стратегия часто теряется среди компонентов плана тестирования. Мы видели документы с планами, в которых подробно описывается график, состав команды и какие документы должны быть получены в ходе тестирования, но почти ничего не говорится о том, как будет тестироваться продукт. Это явная недоработка, из-за чего создается впечатление, что для тестирования достаточно посадить тестировщика за клавиатуру в нужное время и в нужном месте. Не ждите, что ваши коллеги будут уважать то, чем вы занимаетесь как тестировщик, если вы не расскажете им об этом.

Как тестовые сценарии позволяют лгать

Если вы возьмете все портфели в вашей компании и сложите их в кучу, то ничего не узнаете о ценности того, что в них находится. Допустим, у вас 37 портфелей и все вместе они весят 384 фунта. Что это говорит о будущем компании? Ничего. А вот содержимое этих портфелей может иметь самое непосредственное отношение к вашему будущему. Единственный способ решить этот вопрос — открыть портфели и разобраться с их содержимым.

Тестовые сценарии подобны портфелям. Их подсчет без учета содержимого ни о чем не говорит. Точно так же ни о чем не говорит и подсчет соотношения количества пройденных и проваленных тестов: хорошо это или ужасно — 90 % успешных тестов? Никто не может этого знать, не будучи знакомым с содержимым тестов. Подсчет соотношения реализованных и запланированных тестов тоже ни о чем не говорит: может быть, самые сложные были отложены на конец проекта, а реализация последних 10 % потребует 50 % времени. А может быть, количества

запланированных тестов совершенно недостаточно для проверки значимых рисков.

Иногда менеджеры по тестированию соглашаются с нами в этом, но считают, что у них нет альтернативы. Что ж, альтернатива есть: не лгать. Мы считаем, что лучше мало знать и работать, исходя из этого, чем мало знать и делать вид, что знаешь много. Есть и другая альтернатива: поговорить о рисках и покрытии. Другими словами, изучите и обсудите содержимое ваших тестов[1].

Тестировщики, использующие неясные и необъяснимые метрики тестирования, чтобы передать клиенту информацию о степени и полноте тестирования, намеренно или нет *вводят его в заблуждение*.

Стратегия тестирования — нечто большее, чем тесты

Ваша стратегия содержит обоснование проводимого вами тестирования. Когда вас спрашивают о стратегии, может показаться, что лучший ответ — это показать на свои тесты и сказать: «Эти 500 тестов — моя стратегия тестирования этого продукта». Может быть, это и точный ответ, но не очень полезный. Это избыточная информация, и она ничего не говорит о том, насколько хорошо эти тесты позволяют достичь вашей цели. Предоставьте информацию, которая поможет: обобщите методы и мотивы, которые послужили основой ваших тестов.

---

[1] Джоанна Ротман считает, что наша позиция в этом вопросе слишком радикальна. Она пишет: «Другая альтернатива, которая у вас есть, — это обсудить, что означают эти цифры. Например, если коэффициент прохождения тестов падает с 98 до 30 %, а проект только начался, то стоит ли беспокоиться? Скорее всего, нет. Но если через неделю состоится бета-тестирование или, что еще хуже, отправка продукта, то, скорее всего, вам стоит беспокоиться. Почему? В начале проекта вы не ожидаете выполнения многих тестов. В конце — ждете, что большинство тестов (если не все) будут выполняться. Вы можете использовать эти цифры, чтобы обсудить, почему ваши ожидания не оправдываются. Возможно, вам не хватает людей для выполнения всех тестов, или для написания баг-репортов, или для проверки исправленных багов. Вы можете использовать эти цифры, чтобы объяснить свои опасения».

Джефф Блейберг пишет: «Как правило, возникает необходимость обеспечить видимость процесса тестирования. Это можно сделать с помощью метрик. Но я убедился, что независимо от того, насколько “надежными” и “понятными” мне кажутся метрики, люди будут понимать их неправильно. Поэтому одним из моих правил является то, что я никогда не раздаю отчеты, а провожу совещания, на которых обсуждаю метрики и их значение».

Ваша стратегия объясняет суть тестирования

Когда у вас есть четкая стратегия и вы проводите тестирование в соответствии с ней, вы можете быстро и доходчиво объяснить процесс тестирования любому, кто об этом спросит. Это позволяет получить поддержку процесса тестирования со стороны остальных членов команды. Если вы не определились со стратегией тестирования, то ваши объяснения, скорее всего, будут звучать путано и неубедительно.

Хорошей является стратегия тестирования, которая обладает следующими качествами.

- *Ориентированность на конкретный продукт.* Общая стратегия может быть очень хорошей, но стратегия, предназначенная для конкретного продукта и технологии, будет куда более результативной.
- *Ориентированность на риск.* Покажите, как в процессе тестирования будут решаться наиболее важные задачи. Свяжите процесс тестирования с вашей целью в этом проекте.
- *Диверсифицированность.* В большинстве ситуаций диверсифицированная стратегия лучше, чем монолитная. Диверсифицированной является стратегия, содержащая множество различных техник или подходов к тестированию. Проблемы, которые могут быть пропущены в случае использования одного подхода к тестированию, вполне могут быть выявлены с помощью другого.
- *Практичность.* Стратегия тестирования должна быть осуществимой. Не предлагайте стратегию, которая выходит далеко за рамки возможностей проекта.

Применяйте различные полумеры

Менее глубокая и более диверсифицированная стратегия тестирования лучше, чем стратегия, обладающая обратными свойствами. Другими словами, лучше провести больше различных видов тестирования на достаточно хорошем уровне, чем идеально провести один или два вида тестирования. Мы называем это принципом *разнообразных полумер*.

Этот стратегический принцип вытекает из структурной сложности программных продуктов. При тестировании выбирается сложное пространство. Ни одна техника тестирования не позволяет выбрать это

пространство таким образом, чтобы быстро обнаружить все важные проблемы. Любая техника сначала дает возможность найти много багов, но затем кривая скорости их обнаружения выровняется. Если вы перейдете на технику, ориентированную на другой тип проблем, то скорость поиска может снова возрасти. С точки зрения общей продуктивности поиска багов используйте каждую технику до момента, пока результат ее применения не станет достаточно низким, и переходите к новой технике.

Диверсификация преследует еще одну важную цель: не оказаться в ситуации, когда после многомесячного тестирования продукта и выпуска его на рынок пользователи тут же обнаруживают серьезные проблемы. Такая ситуация может быть вызвана несколькими причинами. Основная — туннельное зрение. Дело не в том, что вы тестировали недостаточно, а в том, что вы не проводили правильные тесты. Мы сталкивались со случаями, когда компания проводила сотни тысяч тестов и все равно не замечала простых очевидных проблем, поскольку проводила недостаточное количество тестов.

Для обеспечения разнообразия используйте пятимерную систему тестирования (см. главу 3), чтобы выбрать техники из всех пяти категорий. Диверсифицируйте, чтобы максимизировать количество обнаруженных багов. Диверсифицируйте, чтобы свести к минимуму вероятность пропуска важной проблемы.

Развивайте компетенции и расширяйте ресурсы, позволяющие реализовать эффективные стратегии тестирования

Реализация стратегии становится возможной благодаря соответствующим ресурсам. На протяжении текущего проекта, а также в ходе всех остальных развивайте эти ресурсы, чтобы максимально использовать стратегические возможности. К числу таких ресурсов относятся:

- ваш уровень владения каждой техникой тестирования;
- знание технологий, лежащих в основе продукта;
- друзья, обладающие специальными навыками тестирования или техническими навыками;
- хранилища исходных тестовых данных;
- разнообразие тестовых платформ, в том числе несколько операционных систем и аппаратных конфигураций;

- разнообразные средства тестирования;
- фактические данные пользователя;
- функции тестируемости, встроенные в продукт (например, файлы журнала, утверждения и тестовые меню).

Ваша первая стратегия в проекте всегда неверна

Стратегия должна меняться по мере изучения продукта и его модели отказов. Мы рекомендуем основывать стратегии на рисках. При этом возникает проблема: *вы не знаете, что такое риски*. В начале проекта все, что у вас есть, — это слухи о том, в какой части продукта присутствуют серьезные баги. Если повезет, то будут и обоснованные догадки. Поэтому ваша стратегия в начале проекта, скорее всего, будет подвержена по крайней мере одной из двух проблем: вы не ориентированы на риски или ориентированы на области, которые кажутся рисками, но таковыми не являются.

Решить эту проблему можно, если не принимать преждевременных решений о выборе исключительно одной стратегии тестирования. Пусть ваша стратегия меняется по мере того, как вы узнаете больше информации о продукте, увидите его слабые стороны и придумаете новые способы тестирования.

Такие жизненные циклы проектов, как V-модель, предполагают, что вы будете знать, какой должна быть ваша стратегия в самом начале проекта. Для того чтобы с первого раза правильно определить стратегию, нужно быть не просто гением, а ясновидящим, поэтому мы считаем, что V-модель — плохой способ организации проекта. Если вы решили использовать V-модель, то рекомендуем разработать тест-план на ранних этапах, а новые и более совершенные тесты добавлять на более поздних, когда все перестанут обращать внимание на документацию и будут просто пытаться отгрузить хороший продукт в срок.

На каждом этапе проекта спрашивайте себя: «Что я могу протестировать сейчас и как я могу сделать это?»

Иногда мы видим проекты, в которых берется фаза проекта, например системная интеграция, и заявляется, что в ходе этой фазы допустимы

только определенные виды тестирования, например тестирование на основе требований. Это вызывает такое же недоумение, как и запрет на ношение белого цвета после Дня труда[1]. Зачем следовать таким общим и огульным правилам?

Фаза проекта, во время которой проводится тестирование, и архитектурный уровень, на котором оно проводится (модуль, подсистема или система), — факторы, определяющие стратегию тестирования, но не доминирующие. Мы рекомендуем в любой момент реализации проекта просто задавать себе вопрос: «Что я могу протестировать здесь и сейчас и как сделать это хорошо?»

Не стоит полагать, что определенные техники полезны только в определенное время. Пусть ваша стратегия тестирования будет ситуативной. В любое время тестируйте все, что стоит тестировать, и используйте те техники, которые лучше всего подходят для обслуживания ваших клиентов.

Тест на зрелость продукта

Хотя общие стратегии тестирования фаз проекта нерациональны, имеет смысл проводить различные виды тестирования в зависимости от степени готовности продукта (рис. 11.2).

- *При тестировании на ранних этапах проекта относитесь к багам терпимее.* В этот период продукт работает не очень хорошо и вы не очень много о нем знаете. Жесткие тесты на данной стадии по большому счету не нужны, поскольку даже простые тесты найдут баги. Кроме того, программисты, которые знают, что продукт еще не готов, нервничают из-за того, что тестировщики слишком строго смотрят на их детище. Программисты хотят знать, работают ли функции в том виде, в котором они реализованы, в принципе.

[1] День труда (англ. Labor Day) — национальный праздник в США, отмечаемый в первый понедельник сентября. Долгое время существовало негласное правило, запрещающее надевать белое после этого праздника. По одной версии, правило возникло в конце XIX в. среди состоятельных американцев. В их среде носить белое после Дня труда считалось пошлостью. Следуя этому правилу, они демонстрировали свое отличие от нуворишей (людей, которые разбогатели недавно, а не воспользовались семейным состоянием), не обладавших знаниями этикета. По другой версии, отказ от белого цвета означал переход на зимний гардероб. — *Примеч. ред.*

Рис. 11.2. Жизненный цикл продукта

- *В середине проекта тестируйте агрессивно.* По мере того как части продукта собираются воедино, реализуются и утрясаются основные функции, простые тесты теряют свою эффективность. Программисты также чувствуют себя более уверенно по отношению к продукту. Они переходят от разработки и реализации функций к полноценному исправлению багов. Именно в это время можно использовать тесты с большей сложностью и более высокими требованиями. Выявляйте нестабильные части продукта по максимуму. Найдите как можно больше багов и сообщите о них. Создайте резервную копию багов, которые разработчики должны исправить.
- *Ближе к концу проекта проводите разнообразные виды тестирования.* В зрелом продукте баги найти сложнее, поэтому приходится проявлять больше изобретательности. В это время вы можете проводить тестирование самыми разнообразными способами, в зависимости от вашего воображения и той поддержки, которую вам оказывает руководство. Используйте помощников, автоматизацию, специальные мероприятия по тестированию (вечеринки по поиску

багов или «баунти»[1]), эвристику, бета-тестировщиков — все что угодно. Если вы все сделаете правильно, то ваш коэффициент обнаружения багов будет очень похож на идеальную кривую на диаграмме. Поднимите его до высокого уровня, используя агрессивное тестирование, и поддерживайте, диверсифицируя и редиверсифицируя тесты, пока у вас просто не закончатся идеи для новых и лучших тестов.

- *В последние дни тестируйте скрупулезно.* Ошибка, допущенная в последние дни, может дорого обойтись вашей компании. По мере приближения даты отгрузки тестирование должно стать более защитным. Тщательно тестируйте каждое изменение. Убедитесь, что все выпускаемые файлы имеют правильную версию. Используйте парное тестирование, чтобы каждый тест мог провести еще один человек.

Идеальный результат тестирования показан на рис. 11.3.

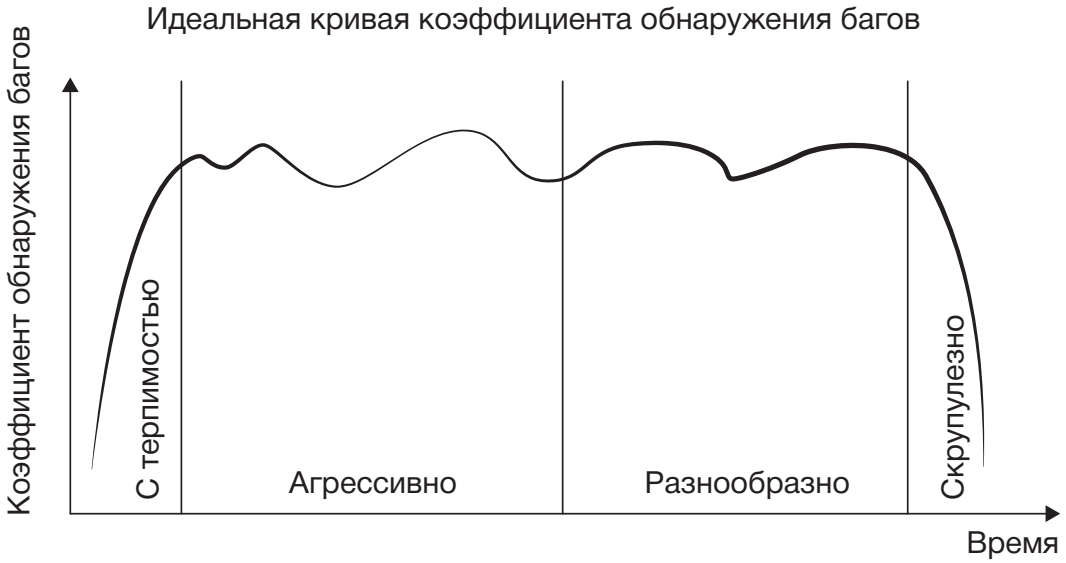

Рис. 11.3. Идеальный результат длительного агрессивного тестирования

[1] Bug Bounty — программа, предлагаемая некоторыми сайтами и разработчиками ПО, с помощью которой люди могут получить признание и вознаграждение за нахождение багов, особенно тех, которые касаются эксплойтов и уязвимостей. Эти программы позволяют разработчикам обнаружить и устранить баги до того, как о них узнает широкая общественность, и таким образом предотвратить злоупотребления. — *Примеч. ред.*

Общая цель — постоянно корректировать стратегию по мере совершенствования продукта, чтобы скорость обнаружения важных багов была высокой на протяжении всего проекта.

Используйте уровни тестирования, чтобы упростить обсуждение сложности теста

Чтобы упростить общение, связанное со сложностью тестов, входящих в стратегию, многие тестовые проекты считают полезным различать уровни тестирования. На низших уровнях проводят более простые и менее мощные тесты, на высших — более сложные. Такое деление может упростить обсуждение стратегии тестирования, поскольку позволяет говорить о классах тестов. Приведем один из примеров иерархии уровней тестирования.

- *Уровень 0, смоук-тестирование.* Простые тесты, которые показывают, что продукт готов к независимому тестированию; проверка работоспособности. Если тесты уровня 0 не работают, то отправьте продукт обратно программистам.
- *Уровень 1, тестирование возможностей.* Тесты, проверяющие возможности каждой функции изделия. Цель — убедиться в том, что каждая функция способна выполнять свою задачу. На этом уровне не допускаются запутанные сценарии, сложные данные и взаимодействие функций.
- *Уровень 2, функциональное тестирование.* В ходе тестов проверяются как возможности, так и базовая надежность каждой отдельной функции и подфункции продукта. Представляют интерес методы покрытия данных и комплексной оценки результатов тестирования. Используйте граничные, стресс-тесты и тесты на обработку ошибок; избегайте сложных сценариев и взаимодействия функций.
- *Уровень 3, сложные тестовые сценарии.* Тестовые сценарии, проверяющие взаимодействия и потоки управления между группами функций. Фокус оценки расширяется и может охватывать оценку производительности, совместимости, нехватки ресурсов, утечек памяти, долговременной надежности или других критериев качества, которые становятся проверяемыми, когда продукт становится более зрелым.

Каждому из этих уровней может соответствовать несколько различных техник тестирования или их сочетаний. Общий смысл заключается в том, чтобы начать тестирование с широких и либеральных позиций,

а затем, по мере развития продукта, переходить к более глубокому и изощренному тестированию. Выполнение тестов третьего уровня на очень ранней сборке продукта, без предварительного выполнения тестов первого и второго уровней, скорее всего, приведет к тому, что программисты будут очень раздражены. Еще более вероятно, что вы не сможете выполнить эти тесты.

Тестируйте методом «серого ящика»

Даже если вы, скорее всего, не имеете всей информации о внутреннем устройстве тестируемого продукта, стратегия тестирования, частично основанная на внутреннем устройстве, может быть весьма эффективной. Мы называем это тестированием методом «серого ящика». Концепция проста: если вы знаете что-то о внутреннем устройстве продукта, то можете лучше протестировать его снаружи. Не следует путать это с тестированием методом «белого ящика», при котором подробно рассматриваются внутренние компоненты продукта. Методом «серого ящика» вы тестируете продукт снаружи, как и методом «черного ящика», но при этом ваши решения по тестированию основываются на знании того, как работают и взаимодействуют базовые компоненты.

Тестирование методом «серого ящика» особенно важно для сетевых и интернет-приложений, поскольку Интернет построен на слабо интегрированных компонентах, которые соединяются через относительно четко определенные интерфейсы. Если вы не понимаете архитектуру Сети, то ваше тестирование будет поверхностным. В книге Хунга Нгуена *Testing Applications on the Web* (Nguyen, 2000) описан хороший пример стратегии тестирования методом «серого ящика», применяемой в Сети.

Остерегайтесь культа предшественников при повторном использовании тестовых материалов

При повторном использовании теста или любого другого тестового материала не относитесь к нему как к «черному ящику». Познакомьтесь с ним. Изучите его. Выясните, почему он разработан именно так. Во многих случаях архивные тестовые материалы имеют настолько низкое качество, что лучше не использовать их повторно. Многие тестировщики склонны доверять тестовым материалам только потому, что те *являются* старыми и, возможно, загадочными. Мы неоднократно обнаруживали, что тесты

регулярно выполняются спустя долгое время (в некоторых случаях — спустя годы) после того, как они устарели. Мы видели, как опытные тестировщики беспрекословно выполняли тесты, написанные гораздо менее опытными тестировщиками. Это темная сторона повторного использования.

В одной из известных нам лабораторий тестирования действует постоянное правило: «Всегда тестировать на ноутбуках Compaq Presario[1], печально известных своей несовместимостью». Странно, но это правило было написано в 1996 году. Актуально ли оно сегодня? Возвращался ли кто-нибудь к этому вопросу? Осмелится ли кто-нибудь усомниться в авторитете Древних, написавших это правило?

Если идея написания тестов, которые будут использоваться повторно, кажется вам привлекательной, то спросите себя, как люди узнают, что означают ваши тесты, зачем они существуют и когда от них следует отказаться ввиду устаревания или изобрести заново. В противном случае велика вероятность того, что клиенты будут относиться к вашей работе как к магическому тотему, а не использовать ваши тесты по назначению.

Два тестировщика, работающие с одним и тем же продуктом, скорее всего, совершают разные действия

Некоторые тестировщики могут впадать в отчаяние из-за возможности дублирования действий. Расслабьтесь. Не стоит беспокоиться о том, что люди будут повторять действия друг друга при выполнении одних и тех же тестовых заданий или тестировать один и тот же продукт одним и тем же способом. Дублирование действий по тестированию не то же самое, что дублирование тестов. Существует довольно большая вероятность, что два тестировщика, работающие с одним и тем же продуктом, могут найти разные проблемы. Это связано с тем, что тестировщики, скорее всего, не выполняют одни и те же тесты. Даже если кажется, что тесты дублируются, между ними обязательно будут различия. Кроме того, один тестировщик может заметить проблему, которую другой не заметил. Это очень частое явление.

Дублирование работы по тестированию практически никогда не является напрасным. На самом деле вопрос заключается не в тратах, а в том, заслуживает ли та или иная часть продукта двойного тестирования.

---

[1] Линейка потребительских настольных компьютеров и ноутбуков, выпускавшихся в США в 1990-е гг.

Разрабатывайте стратегию тестирования с учетом факторов проекта, а также рисков продукта

Хорошая стратегия формируется под влиянием не только рисков продукта, но и факторов внутри проекта. Некоторые стратегические принципы, основанные на проекте, указаны ниже.

- *Не позволяйте багам теряться на границах.* Не применяя разнообразные полумеры и не перекрывая задания по тестированию, вы рискуете не протестировать что-то из-за того, что оно лежит на границе между заданиями двух тестировщиков (или команд).
- *Тестируйте то, что вас просят проверить.* Вы проводите тестирование по поручению многих клиентов. Что, по их мнению, вы должны проверять? Выясните это и убедитесь, что делаете хотя бы часть из этого.
- *Время от времени проверяйте то, что вас попросили не тестировать.* Иногда вас просят не тестировать определенные части продукта. Это может быть очень деликатным вопросом, и мы не можем указывать вам, что делать. Однако иногда именно те вещи, которые просят не тестировать, больше всего нуждаются в этом.
- *Тестируйте запутанные и конфликтные элементы.* Там, где есть путаница и конфликты, баги — обычное дело. Если программист, похоже, не совсем уверен в том, что должна делать та или иная функция, то протестируйте ее. Если он новичок в технологии, то протестируйте ее. Если два программиста создают устройства, взаимодействующие друг с другом, то протестируйте интерфейс. Вы не будете разочарованы.
- *Не тратьте время на «мертвую» функцию.* Когда становится ясно, что функция полна багов, не продолжайте ее тестирование до тех пор, пока не свяжетесь с разработчиком. Причина может быть в плохой сборке или плохой конфигурации. Кроме того, если компонент настолько плох, что его собираются заменить, а не доработать, то все найденные вами баги будут закрыты, поэтому не утруждайте себя тестированием.
- *Больше изменений — больше тестов.* Теоретически малейшее изменение в продукте может привести к большим и нелокальным эффектам. Это означает, что любое изменение может свести на нет все результаты тестирования продукта. В реальности большинство изменений имеют достаточно локальный эффект. Тем не менее, безусловно, верно, что за изменениями необходимо следить

с помощью тестирования. Чем больше их в продукте, тем больше тестов нужно проводить. Этот момент становится очень важным в конце проекта.

Рассматривайте циклы тестирования как пульсацию процесса тестирования

Тестирование осуществляется циклами. Они начинаются со сборки и заканчиваются одним из двух моментов: следующей сборкой или решением о том, что дальнейшее тестирование нецелесообразно. Стратегия тестирования становится более конкретной в ходе циклов тестирования. Планируйте их таким образом, чтобы как можно быстрее предоставлять клиентам полную информацию. Один из способов организации цикла тестирования выглядит следующим образом.

1. *Получите продукт.* Убедитесь, что получили правильную сборку.
2. *Настройте систему тестирования.* Очистите свою систему тестирования. Восстановите образ диска до первозданного состояния или полностью удалите предыдущую версию продукта.
3. *Проверьте тестируемость.* Хорошая ли это сборка? Стоит ли тратить время на тестирование? Начните с проведения смоук-тестов: простых тестов, которые демонстрируют, что каждая основная функция представлена и в принципе работает.
4. *Определите, что является новым или измененным.* Какой новый код, расширяющий или изменяющий возможности продукта, был написан?
5. *Определите, какие баги были исправлены.* Вдобавок обратите внимание на проблемы, которые были отклонены, и отреагируйте на них соответствующим образом.
6. *Тестируйте исправления.* Тестируйте их в первую очередь, пока они еще свежи в памяти программистов. Если исправления не работают, то желательно, чтобы программисты узнавали об этом как можно скорее.
7. *Тестируйте новые или измененные области.* Следующая самая горячая тема в умах программистов — это, пожалуй, новый код.
8. *Протестируйте другие области (помните, что сначала нужно протестировать то, что связано с повышенным риском).* Теперь тестируйте все остальное (что имеет значение), пока не закончится время или пока не протестируете все в достаточной степени.

Если у вас есть автоматизированные регрессионные тесты, то запустите их.

9. *Сообщите о результатах.* Это следует делать периодически, не реже одного раза в день, на протяжении всего цикла тестирования.

КАК РАЗРАБОТАТЬ КОНТЕКСТНО ОРИЕНТИРОВАННЫЙ ТЕСТ-ПЛАН

Данное руководство поможет вам при планировании тестирования. Помните, что реальный тест-план — это набор идей, фактически направляющих ваш процесс тестирования. Мы разработали это руководство таким образом, чтобы оно было полезно независимо от того, составляете вы *документ* плана тестирования или нет.

Это не шаблон. Это не форма, требующая заполнения. Это набор идей, призванных подтолкнуть вас к размышлениям, чтобы вы реже забывали о чем-то важном. Мы используем резкие формулировки и описания, и они могут оказаться неподходящими для начинающего тестировщика. Данное руководство предназначено скорее для поддержки опытного тестировщика или руководителя тестирования.

Ниже приведены семь тем задач. Изучайте темы в любом порядке. Можете свободно переходить от одной к другой. Просто поймите, что качество вашего тест-плана зависит от того, насколько хорошо вы выполнили задания и рассмотрели вопросы, подобные описанным здесь. Врезки под названием *«Проверка состояния»* помогут определить, когда вы составили достаточно хороший план, но мы рекомендуем снова проходить по пунктам и пересматривать ваш план (по крайней мере в голове) на протяжении всего проекта.

Тема 1. Мониторинг основных проблем, связанных с планированием тестирования

Ищите риски, препятствия или другие проблемы, оказывающие влияние на время, усилия или возможность планирования практической и эффективной стратегии тестирования. Получите представление об общем объеме работ по планированию. Контролируйте эти вопросы на протяжении всего проекта.

ПРОВЕРКА СОСТОЯНИЯ

- Являются ли какие-либо стандарты качества продукции особенно важными для достижения или трудноизмеримыми?
- Является ли продукт сложным или трудным для изучения?
- Потребуются ли тестировщикам специальная подготовка или инструменты?
- Сложно ли получить или сконфигурировать какую-либо часть тестовой среды?
- Будете ли вы тестировать неинтегрированные или наполовину работающие компоненты продукта?
- Существуют ли какие-либо особые проблемы с тестируемостью?
- Хватает ли команде проекта опыта работы с дизайном продукта, технологией или базой пользователей?
- Должно ли тестирование начаться как можно быстрее?
- Отсутствует ли еще какая-либо информация, необходимая для планирования?
- Можете ли вы ознакомиться с тестируемой версией продукта (даже демонстрационной, прототипом или старой версией)?
- Трудно ли нанять компетентный персонал по тестированию или организовать его работу?
- Нужно ли будет придерживаться незнакомой методологии тестирования?
- Учитываются ли при составлении планов проекта потребности тестирования?
- Подлежит ли план длительному согласованию или утверждению?
- Вы находитесь вдали от своих клиентов (сотрудничаете удаленно)?
- Часто ли меняются планы проекта?
- Будет ли план подвергаться аудиту?
- Ваши клиенты не знают, чего хотят от вас?

Тема 2. Уточнение цели

Все цели, приведенные в этом разделе (или любая из них), могут быть частью вашей цели тестирования. Некоторые из них более важны.

Основываясь на своих знаниях о проекте, выполните ранжирование этих целей. Если это применимо, то приведите конкретные показатели успеха, по которым вас будут оценивать.

Элементы цели, которые необходимо учитывать

- Быстрый поиск важных проблем.
- Выполнение комплексной оценки качества.
- Сертификация качества продукции на соответствие определенному стандарту.
- Минимизация времени или стоимости тестирования.
- Максимальное повышение эффективности тестирования.
- Консультирование клиентов по вопросам повышения качества или тестируемости.
- Консультирование клиентов по вопросам проведения тестирования.
- Необходимость убедиться, что процесс тестирования полностью подотчетен.
- Неукоснительное следование определенным методам или инструкциям.
- Удовлетворение запросов конкретных стейкхолдеров.

Возможные результаты работы

- Краткое электронное письмо с описанием вашей цели.
- Одностраничный устав тестового проекта.

ПРОВЕРКА СОСТОЯНИЯ

- Знаете ли вы, кто ваши клиенты?
- Согласны ли с вашей целью те, кому она важна?
- Достаточно ли ясна ваша цель, чтобы вы могли основывать на ней свое планирование?

Тема 3. Анализ продукта

Ознакомьтесь с продуктом и технологией, лежащей в его основе. Узнайте, как будет использоваться продукт. Погрузитесь в него с головой. По мере продвижения проекта ваше тестирование будет становиться все более качественным, поскольку вы будете больше разбираться в продукте.

Что следует анализировать

- Пользователей (кто они и что делают).
- Структуру (код, файлы и т. д.).
- Функции (что делает продукт).
- Данные (входные, выходные, состояния и т. д.).
- Платформы (внешнее аппаратное и программное обеспечение).
- Операции (для чего используется продукт).

Методы анализа

- Проведение исследовательского тестирования.
- Ревью документации по продукту и проекту.
- Опрос разработчиков и пользователей.
- Сравнение с аналогичными продуктами.

Возможные результаты работы

- Схема тестового покрытия.
- Спецификации с примечаниями.
- Перечень проблем продукта.

ПРОВЕРКА СОСТОЯНИЯ

- Одобряют ли тест-дизайнеры схему покрытия продукта?
- Считают ли дизайнеры, что вы понимаете продукт?
- Можете ли вы визуализировать продукт и предсказать его поведение?
- Умеете ли вы создавать тестовые данные (входные и итоговые)?
- Можете ли вы настроить и эксплуатировать продукт?
- Понимаете ли вы, как будет использоваться продукт?
- Известны ли вам пробелы или несоответствия в проекте?
- Нашли ли вы неявные и явные спецификации?

Тема 4. Анализ рисков продукта

По каким причинам этот продукт может оказаться неудачным? Сначала у вас будет в лучшем случае общее представление. По мере продвижения проекта стратегия тестирования и само оно будут улучшаться, поскольку вы узнаете больше о динамике отказов продукта.

Что следует анализировать

- Угрозы (проблемные ситуации и данные).
- Уязвимые места (где возможны нарушения работы).
- Режимы отказов (возможные виды неисправностей).
- Воздействие на жертву (какое значение имеют проблемы).

Методы анализа

- Ревью требований и спецификаций.
- Ревью фактических отказов.
- Опрос разработчиков и пользователей.
- Проверка продукта на соответствие эвристике риска и категориям критериев качества.
- Выявление общих закономерностей возникновения сбоев и отказов.

Возможные результаты работы

- Матрица компонентов/рисков.
- Перечень рисков.

ПРОВЕРКА СОСТОЯНИЯ

- Согласны ли разработчики и пользователи с результатами анализа рисков?
- Сможете ли вы обнаружить все значимые виды проблем, если они возникнут в процессе тестирования?
- Знаете ли вы, какие именно усилия по тестированию позволят достичь максимальной эффективности?
- Могут ли разработчики сделать что-нибудь, чтобы облегчить обнаружение важных проблем или снизить вероятность их возникновения?
- Как определить, насколько точен анализ рисков?

Тема 5. Разработка стратегии тестирования

Что можно сделать для быстрого и эффективного тестирования на основе самой полной информации о продукте? Конечно, лучше всего принимать решения заранее, но пусть ваша стратегия совершенствуется на протяжении всего проекта.

Рассматривайте техники с пяти точек зрения

- Техники, ориентированные на тестировщиков.
- Техники, ориентированные на покрытие (как структурное, так и функциональное).
- Проблемно-ориентированные техники.
- Техники, ориентированные на деятельность.
- Техники, ориентированные на оценку.

Способы планирования

- Соотнесите техники с рисками и областями применения продуктов.
- Визуализируйте конкретные и практические приемы.
- Диверсифицируйте свою стратегию, чтобы свести к минимуму вероятность упускания важной проблемы.
- Ищите способы внедрения автоматизации, позволяющей расширять стратегию.
- Не переусердствуйте с планом. Дайте тестировщикам возможность использовать свои мозги.

Возможные результаты работы

- Подробное изложение каждой выбранной стратегии тестирования и способов ее применения.
- Матрица рисков/задач.
- Перечень вопросов или проблем, присущих выбранным стратегиям.
- Консультация по частям продукта с малым покрытием.
- Тестовые случаи (только при необходимости).

ПРОВЕРКА СОСТОЯНИЯ

- Согласны ли ваши клиенты со стратегией тестирования?
- Все ли ее элементы являются необходимыми?
- Можете ли вы реально реализовать эту стратегию?
- Не является ли стратегия слишком обобщенной и может ли с равным успехом применяться к любому продукту?
- Знаете ли вы о какой-то категории проблем, не покрытых тестами?
- Задействованы ли в стратегии имеющиеся ресурсы и помощники?

Тема 6. Планирование логистики

Как вы будете реализовывать свою стратегию тестирования? На нее большое влияние оказывают логистические ограничения или предписания. Постарайтесь договориться об использовании необходимых вам ресурсов и задействовать все, что у вас уже есть.

Зоны логистики

- Оценка и планирование усилий, затрачиваемых на тестирование.
- Защита тестируемости.
- Укомплектованность команды тестировщиков (необходимая квалификация).
- Обучение и контроль тестировщиков.
- Назначение заданий для тестировщиков.
- Сбор информации о продукте и управление ею.
- Проведение совещаний по проекту, коммуникация и координация.
- Взаимодействие со всеми другими функциями проекта, включая разработку.
- Приобретение и настройка испытательной платформы.
- Соглашения и протоколы.
- Средства тестирования и автоматизации.
- Потребности в создании заглушек и моделировании.
- Управление тестовым пакетом и его сопровождение.
- Сборка и протокол передачи.
- Администрирование цикла тестирования.
- Система и протокол баг-репортов.
- Протокол сообщения о состоянии тестирования.
- Заморозка кода и инкрементное тестирование.
- Управление давлением на разработчиков на завершающей стадии.
- Протокол подписания.
- Оценка эффективности тестирования.

Возможные результаты работы

- Перечень вопросов.
- Анализ рисков проекта.
- Матрица ответственности.
- График тестирования.

ПРОВЕРКА СОСТОЯНИЯ

- Поддерживает ли логистика проекта стратегию тестирования?
- Существуют ли проблемы, препятствующие тестированию?
- Возможна ли адаптация логистики и стратегии на фоне прогнозируемых проблем?
- Можете ли вы начать тестирование сейчас и разобраться с остальными проблемами позже?

Тема 7. Распространение плана

Вы не работаете над проектом в одиночестве. Процесс тестирования должен служить целям проекта. Поэтому вовлекайте участников проекта в процесс планирования тестирования. Не обязательно делать это с размахом. По крайней мере, пообщайтесь с ключевыми членами команды, чтобы узнать их мнение и получить негласное одобрение реализации вашего плана.

Как поделиться информацией

- Привлекайте разработчиков и стейкхолдеров к процессу планирования тестирования.
- Активно запрашивайте мнения по поводу плана тестирования.
- Сделайте все возможное, чтобы помочь разработчикам добиться успеха.
- Помогите разработчикам понять, как их работа влияет на тестирование.
- Поговорите с техническими писателями и сотрудниками технической поддержки об обмене качественной информацией.
- Привлеките дизайнеров и разработчиков к рассмотрению и утверждению справочных материалов.
- Выполняйте регистрацию и отслеживание соглашений.
- Привлекайте людей к рассмотрению частей плана.
- Повышайте удобство рецензирования за счет минимизации лишнего текста в документах плана тестирования.

Цели

- Общее понимание процесса тестирования.
- Общая приверженность процессу тестирования.
- Разумное участие в процессе тестирования.

- Получение руководством обоснованных ожиданий в отношении процесса тестирования.

ПРОВЕРКА СОСТОЯНИЯ

- Уделяет ли команда проекта внимание плану тестирования?
- Понимает ли команда, особенно руководители первого звена, свою роль?
- Чувствуют ли участники проекта, что команда тестировщиков преследует интересы проекта?
- Существуют ли состязательные или конструктивные отношения между командой тестировщиков и остальными участниками проекта?
- Есть ли у кого-нибудь ощущение, что тестировщики «отвлекаются», а не сосредотачиваются на важном тестировании?

Насколько хорош этот тест-план?

Ответ на этот вопрос может быть дан только с учетом представления о том, каким план должен быть. Несмотря на то что в ряде открытых стандартов определены форматы документов тест-планов, они не дают достаточных оснований для того, чтобы отличить лучший план от худшего. Эта модель определяет основные понятия и цели тест-плана, критерии, которым он должен удовлетворять, и некоторые эвристики, помогающие определить, удовлетворяются ли критерии в отношении целей.

Термины и понятия

- *Тест-план.* Набор идей, направляющих или представляющих процесс тестирования. Часто эти идеи документируются лишь частично, распространяются по нескольким документам и могут изменяться по мере развития проекта.
- *Документ тест-плана.* Любой документ, предназначенный для передачи идей плана тестирования. Однако документы тест-плана не единственный источник информации о плане тестирования. Она содержится также в устной традиции проекта и культуре компании.
- *Стратегия тестирования.* Определяет взаимосвязь проекта тестирования и цели тестирования. Касается того, что и как будет тестироваться. Отличается от логистики реализации стратегии.

- *Логистика тестирования.* Средства, с помощью которых реализуется стратегия тестирования и предоставляются результаты. Логистика тестирования содержит в себе такие детали, как кто, где и когда будет проводить тестирование, а также используемые вспомогательные материалы.
- *Процесс тестирования.* Понятие процесса имеет много значений. В данном документе мы используем это слово для обозначения того, как на самом деле происходит тестирование (в отличие от того, как оно должно происходить или как об этом говорится в документах).

Цели тест-плана

Цели тест-плана — это то, что тест-план должен помочь нам сделать. Ниже приведен перечень целей, которых можно достичь благодаря идеальному плану тестирования. Однако в документе тест-плана рассматривается только определенное подмножество этих целей. Остальные описываются в других документах или определяются непосредственно менеджером по тестированию или отдельным тестировщиком без поддержки какого-либо документа. Таким образом, тест-план должен оцениваться только с точки зрения тех целей, которых можно достичь благодаря ему или которые не достигаются с помощью других средств в полном объеме.

- Поддержка развития оценки качества, позволяющей принимать обоснованные и своевременные решения в отношении продукции.
- Описание и обоснование стратегии тестирования с учетом требований к продукту и его рисков. Повышение осведомленности о преимуществах и ограничениях стратегии тестирования.
- Описание и обоснование любых специальных требований или входных критериев, необходимых для продолжения проекта тестирования, а также любого выхода из тестирования или процесса, позволяющего определить момент прекращения тестирования.
- Поддержка начала и организации проекта тестирования, включая подготовку, подбор персонала, делегирование обязанностей, приобретение оборудования, планирование задач и составление графиков.
- Поддержка ежедневного управления проектом тестирования и стратегией тестирования, а также поддержка их оценки.
- Поддержка эффективной координации, сотрудничества и других отношений между членами команды тестировщиков, а также между командой тестировщиков и остальными участниками проекта.
- Выявление любых рисков и проблем, способных повлиять на проект, и управление ими.

- Определение конечных результатов тестового проекта и процесса поставки.
- Запись исторической информации для поддержки аудита процессов, их совершенствования и будущих проектов тестирования.

Критерии качества тест-плана

Насколько хорошо тест-план позволяет достигать указанных целей? Определить это вам может помочь данный список критериев.

- *Полезность.* Будут ли благодаря плану тестирования эффективно достигаться те или иные запланированные цели?
- *Точность.* Является ли документ тест-плана точным в отношении любых утверждений о фактах?
- *Эффективность.* Эффективно ли используются имеющиеся ресурсы?
- *Адаптивность.* Скажутся ли на плане обоснованные изменения проекта?
- *Ясность.* Является ли тест-план внутренне согласованным и достаточно однозначным?
- *Простота использования.* Является ли документ тест-плана кратким, удобным для ведения и грамотно составленным?
- *Соответствие нормативным требованиям.* Соответствует ли тест-план внешним требованиям?
- *Основание плана.* Является ли тест-план результатом эффективного процесса планирования тестирования?
- *Целесообразность.* Соответствует ли тест-план возможностям организации, которая должна его использовать?

Эвристика тест-плана

Для оценки тест-плана необходимо рассмотреть, насколько эффективно он позволяет достигать тех или иных целей и насколько соответствует критериям качества. Для этого мы предлагаем использовать следующие эвристики (табл. 11.1). Эвристика — это эмпирическое правило или обоснованное предположение. Ни одна из эвристик, представленных в таблице, не может считаться одинаково важной в любой ситуации, а некоторые из них могут быть вообще неприменимы в вашем случае. Каждая эвристика описывается в виде общего правила и его краткого обоснования, которое призвано помочь определить, когда и где применяется та или иная эвристика.

Таблица 11.1. Эвристики тест-плана

№	Эвристика	
	Правило	**Обоснование**
1	Быстро решайте важные проблемы. Тестирование должно быть оптимизировано так, чтобы можно было быстро находить важные проблемы, а не пытаться найти все проблемы с одинаковой срочностью	Исправления могут быть сложными и трудоемкими. Они могут привести к появлению новых проблем. Поэтому благодаря максимально раннему обнаружению проблем команда получит наилучшие шансы на их безопасное устранение
2	Сосредоточьтесь на риске. Стратегия тестирования должна концентрировать основные усилия специалистов на областях потенциального технического риска, но при этом часть возможной работы необходимо оставить в областях с низким уровнем риска на случай, если анализ риска окажется неверным	Исчерпывающее тестирование невозможно, и мы никогда не сможем узнать, было ли наше восприятие технического риска абсолютно точным
3	Максимизируйте разнообразие. Стратегия тестирования должна быть диверсифицирована с точки зрения методов и перспектив тестирования. Методы оценки тестового покрытия должны учитывать множество критериев покрытия, таких как структурное, функциональное покрытие, а также покрытие данных, платформы, операций и требований	Ни одна методика тестирования не может линейно выявить все важные проблемы. Никогда нельзя быть уверенными, что все важные проблемы обнаружены. Диверсификация минимизирует риск того, что стратегия тестирования окажется неэффективной по отношению к определенным видам проблем
4	Избегайте излишней детализации тестирования. Избегайте детальной предварительной проработки тестов, если для этого нет особых и веских причин. Стратегия тестирования должна предусматривать разумную вариативность и использовать способность тестировщика с помощью ситуативного мышления концентрироваться на важных, но непредвиденных проблемах	Жесткая стратегия тестирования может повысить вероятность обнаружения определенного подмножества проблем, но в сложной системе она снижает вероятность обнаружения всех важных проблем. Разумная вариативность тестирования, например, в результате интерактивного, исследовательского тестирования увеличивает случайное тестовое покрытие, не нанося существенного ущерба основному покрытию

№	Эвристика	
	Правило	**Обоснование**
5	Проверяйте проект на соответствие замыслу. Важно выполнять проверку на соответствие подразумеваемым требованиям — всего того, что подразумевают требования, а не только того, что в них непосредственно указано. Почему каждое требование важно? Узнайте. Проверяйте дух, а не только букву	Тестирование только по явным письменным требованиям не позволит выявить все важные проблемы, поскольку сформулированные требования, как правило, неполны, а естественный язык по своей природе неоднозначен. Многие требования, вероятно, не записаны
6	Вы не одни. Тест-план должен способствовать взаимодействию со всеми другими подразделениями проекта, особенно с программистами, службой технической поддержки и техническими писателями. По возможности тестировщики должны также сотрудничать с реальными клиентами и пользователями, чтобы лучше понять их требования	Другие команды и стейкхолдеры часто располагают информацией о текущих или потенциальных проблемах продукта. Эти данные могут быть полезны команде тестирования. Эта новая точка зрения способна помочь тестировщикам лучше проанализировать риск. Тестировщики также могут обладать информацией, полезной для них самих
7	Способствуйте тестируемости. Консультируйте программистов, чтобы помочь им создать продукт, более пригодный к тестированию	Вероятность того, что стратегия тестирования выполнит свою задачу, в значительной степени зависит от тестируемости продукта
8	Тест-планы не бывают типовыми. В тест-плане должны быть выделены нерутинные, характерные для проекта аспекты стратегии и проекта тестирования	Каждый программный проект, который стоит реализовать, сопряжен с особыми техническими проблемами, решение которых требует качественного тестирования. Типовой тест-план свидетельствует о слабом процессе планирования тестирования
9	Ничего лишнего. В документе тест-плана следует избегать лишнего текста. Не излагайте очевидные истины. Каждое предложение должно быть значимым. Если вы взаимодействуете с разными аудиториями, то подумайте о подготовке разных версий плана для каждой из них	Любой текст, кажущийся ненужным или очевидным, снижает вероятность прочтения документа. Читатели будут считать, что они уже поняли, о чем идет речь в документе

Продолжение ⇩

Таблица 11.1 *(продолжение)*

№	Эвристика	
	Правило	**Обоснование**
10	Не надо программировать людей. В проекте тестирования люди должны решать свои задачи, а с помощью автоматизации будет выполняться предназначенная для нее работа. Тестирование вручную должно допускать импровизацию и критическое мышление по ситуации, в то время как автоматизированное тестирование следует использовать для тестов, требующих высокой повторяемости, большой скорости и отсутствия предварительных предположений	На многих проектах по тестированию негативно сказывается ложное убеждение, что тестировщики эффективны, если используют точно заданные сценарии тестирования, или что автоматизация тестирования дублирует ценность человеческого познания в процессе выполнения теста. Ручное и автоматизированное тестирование — не две формы одного и того же процесса. Это два совершенно разных класса технологий тестирования
11	Зависимости графика тестирования. Он должен быть представлен и обоснован таким образом, чтобы выделить все зависимости от хода разработки, тестируемости продукта, времени, необходимого для сообщения о проблемах, и оценки риска проектной группой	Монолитный график тестирования в тест-плане часто свидетельствует о ложной уверенности в том, что тестирование представляет собой независимый вид деятельности. График тестирования может быть самостоятельным только в том случае, если продукт хорошо поддается тестированию, разработка завершена, а процесс тестирования не прерывается необходимостью часто сообщать о проблемах
12	Предотвращайте возникновение узких мест. Процесс тестирования должен быть по возможности исключен из критического пути. Этого можно добиться, проводя тестирование параллельно с разработкой и обнаруживая проблемы, требующие устранения в более срочном порядке, нежели это обычно делают программисты	Это важно для того, чтобы снизить давление на разработчиков и позволить сократить процесс тестирования

№	Эвристика	
	Правило	**Обоснование**
13	Обеспечьте быстрое получение обратной связи. Цикл обратной связи между тестировщиками и программистами должен быть как можно более коротким. Циклы тестирования должны быть построены таким образом, чтобы обеспечить быструю обратную связь с программистами по поводу последних дополнений и изменений, внесенных ими до начала полного регрессионного тестирования. По возможности тестировщики и программисты должны работать, находясь рядом друг с другом физически	Это важно для того, чтобы максимально повысить эффективность и скорость улучшения качества. Кроме того, это позволяет исключить тестирование из критического пути
14	Участвовать в тестировании могут не только сами тестировщики. Для оценки и корректировки проекта тестирования следует использовать каналы получения информации о качестве, отличающиеся от формального тестирования. Примерами таких каналов могут служить инспекции, тестирование в условиях эксплуатации или неформальное тестирование, проводимое людьми, не входящими в состав команды тестирования	Изучение информации о качестве продукта, собранной разными людьми, помимо команды тестирования, позволяет выявить «слепые зоны» в формальной стратегии тестирования
15	Рецензирование документации. Вся архивная тестовая документация должна проверяться не только тем, кто ее написал. Используемый процесс рецензирования должен быть соизмерим с важностью документа	Туннельное зрение представляет собой большую профессиональную опасность в сфере тестирования. Рецензирование не только помогает выявить «слепые пятна» в разработке тестов, но и способствует диалогу и взаимному обучению методам тестирования

Приложение

Контекстно ориентированный подход к тестированию программного обеспечения

Мы принадлежим к группе людей, которых иногда называют представителями *контекстно ориентированной школы* тестирования программного обеспечения. По прошествии нескольких лет мы выработали принципы, отражающие, по нашему мнению, общность взглядов людей, осуществляющих интеллектуальное руководство этой школой.

В данной книге представлен большой набор примеров контекстно ориентированного мышления и интерпретации нашего опыта в области разработки ПО. Наряду с книгой мы создали сайт context-driven-testing.com для дальнейшего развития школы.

Если вы прочитали приведенные ниже принципы и описания и решили, что хотите причислить себя к этой школе, то посетите сайт и присоединитесь к сообществу.

СЕМЬ БАЗОВЫХ ПРИНЦИПОВ КОНТЕКСТНО ОРИЕНТИРОВАННОЙ ШКОЛЫ

1. Ценность любой практики зависит от ее контекста.
2. Есть хорошие практики в определенном контексте, но нет лучших.
3. Коллектив сотрудников — самая важная часть контекста любого проекта.

4. Проекты разворачиваются во времени так, что зачастую их масштабы невозможно предсказать.
5. Продукт предоставляет решение. Если проблема не решена, значит, продукт не работает.
6. Качественное тестирование ПО — сложный интеллектуальный процесс.
7. Только благодаря здравому смыслу и мастерству, проявляемым в ходе всего проекта, мы можем достигать нужных результатов в нужное время, чтобы эффективно тестировать наши продукты.

ОПИСАНИЯ ПРИНЦИПОВ В ДЕЙСТВИИ

- Команды тестирования существуют для предоставления услуг, связанных с тестированием. Они не управляют проектом разработки, а обслуживают его.
- Тестирование проводится по поручению стейкхолдеров в процессе разработки, уточнения, отладки, исследования или продажи продукта. Для всех этих разных целей могут подойти совершенно разные стратегии тестирования.
- Совершенно правильно, что у разных команд тестирования разные цели. Основная практика, используемая при достижении одной цели, может оказаться неактуальной или контрпродуктивной при выполнении другой.
- Недостоверные метрики опасны.
- Основная ценность любого тестового случая заключается в его способности предоставлять информацию (то есть уменьшать неопределенность).
- Все оракулы ошибаются. Даже если кажется, что продукт прошел тест, вполне возможно, что он его провалил, а вы (или программа автоматизированного тестирования) не смогли это обнаружить.
- Автоматизированное тестирование не автоматическое ручное тестирование. Нелепо говорить об автоматизированных тестах так, будто это выполняемое человеком автоматизированное тестирование.
- Различные типы дефектов будут выявляться различными типами тестов — по мере повышения стабильности программы тесты должны становиться более сложными или фокусироваться на различных рисках.
- Артефакты тестирования имеют ценность в той степени, в какой удовлетворяют соответствующим требованиям стейкхолдеров.

ПРИМЕР

Рассмотрим два проекта. Один из них — разработка ПО для управления самолетом. Что такое *«правильное поведение»* — вопрос сугубо технический и математический. Необходимо соблюдать правила Федерального управления гражданской авиации. Все, что вы сделаете или не сделаете, станет доказательством в судебном процессе через 20 лет. Сотрудники отдела разработок разделяют основные ценности инженерной культуры. Предпочтение в ней отдается осторожности, точности, повторяемости и перепроверке работы каждого.

Другой проект — разработка текстового процессора, работающего через Интернет. «Правильное поведение» — это то, что привлекает огромную и невзыскательную аудиторию пользователей Microsoft Word к вашей программе. Нет никаких важных нормативных требований (кроме регулирующих публичное размещение акций). При этом важно время выхода на рынок — через 20 месяцев все будет завершено, так или иначе. В отделе разработки явно *не* развита инженерная культура, и попытки говорить с его сотрудниками в стиле, который является нормальным для культуры из первого примера, приведут к тому, что они будут называть вас «проблемой, которую стоит обойти стороной».

Практика тестирования, подходящая для первого проекта, не сработает во втором. Действия, уместные во втором проекте, в первом были бы преступной халатностью.

СОСТАВ КОНТЕКСТНО ОРИЕНТИРОВАННОЙ ШКОЛЫ

Если вы согласны с этими принципами и хотите, чтобы вас называли участником этой школы, то, пожалуйста, напишите нам по адресу context@satisfice.com.

Следующие авторы, работы которых публикуются, заявили о своем согласии с этими принципами:

Кем Кейнер
Джеймс Бах
Брет Петтикорд
Анна С. У. Эллисон
Стеле Амланд
Берни Бергер
Джайя Р. Карл
Росс Коллард
Кристофер Денардис
Мардж Фаррелл
Эрик Гриффин
Сэм Гакенхаймер
Элизабет Хендриксон
Кэти Иберл
Боб Джонсон
Карен Джонсон
Марк Джонсон
Алан А. Йоргенсен, доктор наук
Брайан Марик
Патриция А. Маккуэйд, доктор наук
Алан Мирвольд
Ноэль Найман
Пэт Макги
Джоанна Ротман
Джейн Степак
Пол Шимковяк
Энди Тинкхэм
Стив Толман

Литература

A

Agre P. 2001. Networking on the Network. dlis.gseis.ucla.edu/people/pagre/network.html.

Amland S. 1999. Risk Based Testing and Metrics. www.amland.no/Word%20Documents/EuroSTAR%20'99%20Paper.doc.

Asböck S. 2000. Load testing for eConfidence. Lexington, MA: Segue Software, Inc. www.segue.com.

Association for Computing Machinery. 2000. A Summary of the ACM Position on Software Engineering as a Licensed Engineering Profession. www.acm.org/serving/se_policy/selep_main.html.

Austin R. 1996. Measuring and Managing Performance in Organizations. New York: Dorset House Publishing.

B

Bach J. 1999a. Reframing Requirements Analysis // IEEE Computer, № 32:6. P. 113–114.

Bach J. 1999b. A Low Tech Testing Dashboard. www.satisfice.com/presentations/dashboard.pdf.

Bach J. 1999c. James Bach on Risk-Based Testing // STQE Magazine. Vol. 1 #6.

Baron J. 1994. Thinking and Deciding. Cambridge: Cambridge University Press.

Beck K. 1999. Extreme Programming Explained. Reading, Massachusetts: Addison-Wesley.

Beck K., Beedle M., Bennekum A. van, Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R. C., Mellor S., Schwaber K., Sutherland J., Thomas D. 2001. Manifesto for Agile Software Development [online]. www.agilealliance.org/.

Beizer B. 1990. Software Testing Techniques, 2nd edition. Boston: International Thompson Computer Press.

Bender R. A. 1991. Requirements-based Testing // Quality Assurance Institute Journal, 27–32.

Black, Rex. 1999. Managing the Testing Process. Redmond, Washington: Microsoft Press.

Booth W. C., Colomb G. G., Williams J. M. 1995. The Craft of Research. Chicago: University of Chicago Press.

Brooks F. P. 1995. The Mythical Man-Month: Anniversary Edition with Four New Chapters. Reading Massachusetts: Addison-Wesley.

Brown J. S., Duguid P. 2000. The Social Life of Information. Boston: Harvard Business School Press.

Buwalda H., Kasdorp M. 1999. Getting Automated Testing Under Control, Software Testing & Quality Engineering, November 1999.

C

Chapman J. 1996. Negotiating Your Salary: How to Make $1000 a Minute, 3rd edition. Berkeley: Ten Speed Press.

Cohen D. M., Dalal S. R., Parelius J., Patton G. C. 1996. The Combinatorial Design Approach to Automatic Test Generation // IEEE Software. Volume 13#5, September. www.argreenhouse.com/papers/gcp/AETGissre96.shtml.

Cohen D. M., Dalal S. R., Fredman M. L., Patton G. C. 1997. The AETG System: An Approach to Testing Based on Combinatorial Design // IEEE Transactions on Software Engineering. Vol 23#7, July. www.argreenhouse.com/papers/gcp/AETGieee97.shtml.

Cohen N. 2000. Building a Testpoint Framework // Dr. Dobbs Journal. March 2000.

Collard R. 1999. Deriving Test Cases from Use Cases. Software Testing & Quality Engineering, July — August.

Collard R. 2012. Software Testing & QA Techniques (a multivolume series).

Constantine L. L. 1995. Constantine on Peopleware. Yourdon Press.

Construx 2001. Software Engineering Professionalism Web site. www.construx.com/profession/home.htm.

D

Daconta M. C., Monk E., Keller J. P., Bohnenberger K. 2000. Java Pitfalls: Time-Saving Solutions and Workarounds to Improve Programs. New York: John Wiley & Sons.

De Bono E. 1970. Lateral Thinking: Creativity Step by Step. New York: Harper and Row.

DeMarco T. 1997. The Deadline. New York: Dorset House Publishing. DeMarco, Tom and Timothy Lister. 1999. Peopleware: Productive Projects and Teams, 2nd edition. New York: Dorset House Publishing.

Deming W. E. 1986. Out of the Crisis. Cambridge, Massachusetts: MIT Press.

DeNardis C. 2000. Perspectives of a Test Manager // STQE Magazine 2: 5. https://www.stickyminds.com/?sid=194646&sqry=%2AJ%28MIXED%29%2AR%28relevance%29%2AK%28simplesite%29%25%202AF%28Perspectives%20from%20a%20Test%20Manager%29%2A&sidx=0&sopp=10&ObjectId=1976&Function=DETAILBROWSE&ObjectType=ART.

DiMaggio L. 2000. Looking Under The Hood // STQE Magazine. January.

Dörner D. 1996. The Logic of Failure. Trans. Rita Kimber and Robert Kimber. New York: Metropolitan Books.

Drucker P. 1985. The Effective Executive. Harper Colophon.

Dustin E., Rashka J., Paul J. 1999. Automated Software Testing. Reading, Massachusetts: Addison-Wesley.

Dwyer G., Freeburn G. 1999. Business Object Scenarios: a fifth-generation approach to automated testing // Fewster and Graham.

E

Elmendorf W. R. 1973. Cause-Effect Graphs in Functional Testing Technical Report TR-00.2487, IBM Systems Development Division, Poughkeepsie, N.Y.

F

Fewster M., Graham D. 1999. Software Test Automation: Effective Use of Text Execution Tools. Reading, Massachusetts: Addison-Wesley.

Feynman R. 1989. What Do You Care What Other People Think: Further Adventures of a Curious Character. New York: Bantam Books.

Fisher R., Ertel D. 1995. Getting Ready to Negotiate: The Getting to Yes Workbook.

Fisher R., Ury W., Patton B. 1991. Getting to Yes. Boston: Houghton Mifflin Co.

Freund J. C. 1992. Smart Negotiating: How to Make Good Deals in the Real World. Simon and Schuster.

G

Gause D. C., Weinberg Gerald M. 1989. Exploring Requirements: Quality Before Design. New York: Dorset House Publishing.

Gilb T. 1997. Evo: The Evolutionary Project Managers Handbook. www.result-planning.com/, www.stsc.hill.af.mil/ swtesting/gilb.asp.

Glaser B. G., Strauss A. L. 1999. The Discovery of Grounded Theory: Strategies for Qualitative Research. New York: Aldine de Gruyter.

H

Heidtman Steel Products, Inc. v. Compuware Corp. 1999 U.S. Dist. Lexis 21700, U.S. District Court, N.D. Ohio.

Hendrickson E. 1999. Making the Right Choice: The Features You Need in a GUI Test Automation Tool // STQE Magazine, May. www.qualitytree.com/feature/mtrc.pdf.

Hendrickson E. 2001a. Better Testing, Worse Quality? www.qualitytree.com/feature/btwq.pdf.

Hendrickson E. 2001b. Bug Hunting: Going on a Software Safari. Proceedings of the Software Testing Analysis & Review Conference (STAR East). Orlando, Florida: May.

Hendrickson E. 2001. Bug Hunting. New York: Dorset House Publishing.

Hoffman D. 1999a. Cost Benefits Analysis of Test Automation, Proceedings of the Software Testing Analysis & Review Conference (STAR East). Orlando, Florida: May.

Hoffman D. 1999b. Test Automation Architectures: Planning for Test Automation. Proceedings of the International Software Quality Week. San Francisco, May.

Hoffman D. 2000. The Darker Side of Metrics. Proceedings of the Pacific Northwest Software Quality Conference, October 17–18. Portland, Oregon.

Houlihan P. 2001. Targeted Software Fault Insertion, Proceedings of the Software Testing Analysis & Review Conference (STAR East) Orlando, Florida: May. www.mango.com/technology.

Humphrey W. S. 1990. Managing the Software Process. Reading, Massachusetts: Addison-Wesley.

Humphrey W. S. 1997. Managing Technical People. Reading, Massachusetts: Addison-Wesley.

Hutchins E. 1995. Cognition in the Wild. Cambridge, Massachusetts: MIT Press.

I

IEEE Computer Society. 2001. Software Engineering Body of Knowledge, trial version 0.95. www.swebok.org/documents/stoneman095/ Trial_Version_0_95.pdf.

InstallShield Corporation. 1999. Creating a Project with the NetInstall Spy. http://support.installshield.com/reference/netinstall/UG/ ugchapter3.pdf.

J

Jacobson I. 1992. Object-Oriented Software Engineering. Wokingham: Addison-Wesley.

Jeffries R., Anderson A., Hendrickson Ch. 2000. Extreme Programming Installed. Reading, Massachusetts: Addison-Wesley.

Jensen A. R. 1980. Bias in Mental Testing. Free Press.

Johnson K. 2001. Mining Gold from Server Logs // STQE Magazine. January.

Jorgensen P. C. 1995. Software Testing: A Craftsman's Approach. Boca Raton, Florida: CRC Press.

K

Kaner C. 1995a. Software Negligence and Testing Coverage. Software QA Quarterly volume 2, number 2: 18. http://kaner.com/ coverage.htm.

Kaner C. 1995b. Liability for Defective Documentation. Software QA Quarterly, volume 2, number 3: 8. www.kaner.com/baddocs.htm.

Kaner C. 1996a. Computer Malpractice, Software QA, volume 3, number 4: 23. www.badsoftware.com/malprac.htm.

Kaner C. 1996b. Negotiating Testing Resources. www.kaner.com/negotiate.htm.

Kaner C. 1998a. Avoiding Shelfware: A Manager's View of Automated GUI Testing. www.kaner.com/pdfs/shelfwar.pdf.

Kaner C. 1998b. Liability for Product Incompatibility // Software QA Magazine, September.

Kaner C. 2000a. Measurement of the Extent of Testing. www.kaner.com/pnsqc.html.

Kaner C. 2000b. Architectures of Test Automation. http://kaner.com/testarch.html.

Kaner C., Bach J., Nguyen H. Q., Falk J., Johnson B. 2002. Testing Computer Software, 3rd edition, Volume 1.

Kaner C., Falk J., Nguyen H. 1993. Testing Computer Software, 2nd edition, 1999 reprint. New York: John Wiley & Sons.

Kaner C., Hendrickson E., Smith-Brock J. 2000. Managing the Proportion of Testers to (Other) Developers. Proceedings of the International Software Quality Week. San Francisco.

Kaner C., Pels D. 1997. Article 2B and Software Customer Dissatisfaction. www.badsoftware.com/stats.htm.

Kaner C., Pels D. 1998. Bad Software. New York: John Wiley & Sons.

Kaner C., Vokey J. R. 1984. A Better Random Number Generator for Apple's Floating Point BASIC // Micro, June, 26–35. www.kaner.com/random.html.

Kaplan R. S., Norton D. P. 1996. The Balanced Scorecard: Translating Strategy into Action. Cambridge, Massachusetts: Harvard Business School Press.

Koslowski B. 1996. Theory and Evidence: The Development of Scientific Reasoning. Cambridge, Massachusetts: MIT Press.

Kruchten P. 2000. The Rational Unified Process, an Introduction, 2nd edition. Reading, Massachusetts: Addison-Wesley.

L

Lakatos I. 1976. Proofs and Refutations: The Logic of Mathematical Discovery. Cambridge, Massachusetts: Cambridge University Press.

Lawrence B., Johnson B. 1998. A Product Life Cycle (PLC) Model. www.coyotevalley.com/plc/builder.htm.

Lebow R. 1990. A Journey Into the Heroic Environment. Rocklin, California: Prima Publishing.

Levy D. A. 1997. Tools of Critical Thinking: Metathoughts for Psychology. Boston: Allyn and Bacon.

Linz T., Daigl M. 1998a. GUI Testing Made Painless: Implementation and Results of the ESSI PIE 24306. www.imbus.de/forschung/pie24306/gui_test_made_painless.html.

Linz T., Daigl M. 1998b. How to Automate Testing of Graphical User Interfaces. www.imbus.de/forschung/pie24306/gui/aquis-full_paper-1.3.html.

M

Marick B. 1995. The Craft of Software Testing. Upper Saddle River, New Jersey: Prentice Hall.

Marick B. 1998. When Should a Test be Automated? www.testing.com/writings/automate.pdf.

Marick B. 1999. How to Misuse Code Coverage. www.testing.com/writings/coverage.pdf.

Marick B. 2000. Using Ring Buffer Logging to Help Find Bugs. http://visibleworkings.com/trace/Documentation/ring-buffer.pdf.

Marick B. Undated. How Many Bugs Do Regression Tests Find? www.testingcraft.com/regression-test-bugs.html.

Mead N. 2001. Issues in Licensing and Certification of Software Engineers. July 23. t www.sei.cmu.edu/staff/nrm/license.html.

Michalko M. 1991. Thinkertoys: A Handbook of Business Creativity. Berkeley, California: Ten Speed Press.

Miller L. J. 1998. Get More Money on Your Next Job: 25 Proven Strategies for Getting More Money, Better Benefits, & Greater Job Security. McGraw-Hill Professional Publishing.

N

Nguyen H. Q. 2000. Testing Applications on the Web. New York: John Wiley & Sons.

Norman D. A. 1993. Things that make us smart: Defending human attributes in the age of the machine. Reading, Massachusetts: Addison-Wesley.

Notkin D. A., Gorlick M., Shaw M. 2000. An Assessment of Software Engineering Body of Knowledge Efforts: A Report to the ACM Council. www.acm.org/serving/se_policy/bok_assessment.pdf.

Nyman N. 2000. Using Monkey Test Tools // Software Testing & Quality Engineering, January.

O

Olve N.s-G., Roy J., Wetter M. 1999. Performance Drivers: A Practical Guide to Using the Balanced Scorecard. New York: John Wiley & Sons.

O'Malley M. 1998. Are You Paid What You're Worth? Broadway Books.

Ostrand T. J., Balcer M. J. 1988. The Category-Partition Method for Specifying and Generating Functional Tests // Communications of the ACM. Volume 31 #6: June. P. 676–686.

P

Park S. K., Miller K. W. 1988. Random Number Generators: Good ones are hard to find // Communications of the ACM. October, Volume 31, issue 10. P. 1192–1201.

Pettichord B. 1996. Success with Test Automation. Proceedings of the International Software Quality Week. San Francisco, California: May 1996. www.io.com/~wazmo/succpap.htm.

Pettichord B. 1999. Seven Steps to Test Automation Success. Proceedings of the Software Testing Analysis & Review Conference (STAR West). San Jose, California. November. www.io.com/~wazmo/papers/seven_steps.html.

Pettichord B. 2000a. Beyond the Bug Battle, Proceedings of the Software Testing Analysis & Review Conference (STAR East). Orlando, Florida: May.

Pettichord B. 2000b. Testers and Developers Think Differently // Software Testing & Quality Engineering. January. www.io.com/~wazmo/ papers/testers_and_developers.pdf.

Pettichord B. 2001a. Hey Vendors, Give Us Real Scripting Languages. Stickyminds.com. http://stickyminds.com/sitewide.asp?sid=409206&sqry=%2AJ%28MIXED%2

9%2AR%28createdate%29%2AK%28simplesite%29%2AF%28scripting+languages%29%2A&sidx=0&sopp=10&ObjectId=2326&Function=DETAILBROWSE&ObjectType=COL.

Pettichord B. 2001b. What you don't know may help you. Stickyminds.com. July. www.stickyminds.com/sitewide.asp?ObjectId=2629&ObjectType=COL&Function=edetail.

Pettichord B. 2001c. Let observation be your crystal ball. Stickyminds.com, May. www.stickyminds.com/sitewide.asp?ObjectId=2498&ObjectType=COL&Function=edetail.

Polya G. 1957. How to Solve It. Princeton: Princeton University Press. Popper, Karl. 1989. Conjectures and Refutations: The Growth of Scientific Knowledge. London: Routledge.

PowerQuest Corporation. 2001. Drive Image. www.powerquest.com/driveimage/.

R

Rational Software Corporation. 2001. Rational Purify for Unix. www.rational.com/products/purify_unix/index.jsp.

Rational Software Corporation. 2001. Rational Test Foundation for Windows 2000. www.rational.com/products/testfoundation/ w2k_ds.jsp.

Robinson H. 1999. Finite State Model-Based Testing on a Shoestring. Star West 1999. www.geocities.com/model_based_testing/ shoestring.htm.

S

Schneier B. 2000a. Computer Security: Will We Ever Learn? // Crypto-Gram, May 15. www.counterpane.com/crypto-gram-0005.html.

Schneier B. 2000b. Secrets & Lies. New York: John Wiley & Sons.

Simmonds E. 2000. When Will We Be Done Testing? Software Defect Arrival // Proceedings of the Pacific Northwest Software Quality Conference, October 17–18. Portland, Oregon.

Sims H. P. Jr., Manz C. C. 1996. Company of Heroes. New York: John Wiley & Sons.

Solow D. 1990. How to Read and Do Proofs. New York: John Wiley & Sons.

Strauss A., Corbin J. eds. 1997. Grounded Theory in Practice. Thousand Oaks: SAGE Publications.

Strauss A., Corbin J. 1998. Basics of Qualitative Research, 2nd edition. Thousand Oaks: SAGE Publications.

Sweeney M. R. 2001. Automation Testing Using Visual Basic. Berkeley, CA: Apress.

T

Tarrant J. J. 1997. Perks & Parachutes: Negotiating Your Best Possible Employment Deal, from Salary and Bonus to Benefits and Protection. Times Books.

Telles M., Hsieh Y. 2001. The Science of Debugging. Scottsdale, Arizona: Coriolis.

Tukey J. W. 1977. Exploratory Data Analysis. Reading, Massachusetts: Addison-Wesley.

W

Webster B. 1995. Pitfalls of Object-Oriented Development. M&T Books.

Weick K. E. 1995. Sensemaking in Organizations. Thousand Oaks: SAGE Publications.

Weinberg G. M. 1992. Quality Software Management, Volume 1: Systems Thinking. New York: Dorset House Publishing.

Weinberg G. M. 1997a. Quality Software Management, Volume 2: First- Order Measurement. New York: Dorset House Publishing.

Weinberg G. M. 1997b. Quality Software Management, Volume 3: Congruent Action. New York: Dorset House Publishing.

Weinberg G. M. 1997c. Quality Software Management, Volume 4: Anticipating Change. New York: Dorset House Publishing.

Weinberg G. M. 1998. The Psychology of Computer Programming, Silver Anniversary Edition. New York: Dorset House Publishing.

Weinberg G. M. 2001. An Introduction to General Systems Thinking: Silver Anniversary Edition. New York: Dorset House Publishing.

Whittaker J., Jorgensen A. 1999. Why Software Fails // ACM Software Engineering Notes, July. http://se.fit.edu/papers/SwFails.pdf.

Whittaker J., Jorgensen A. 2000. How to Break Software. Proceedings of the Software Testing Analysis & Review Conference, May. Orlando, Florida.

Whittaker J. 2002. How to Break Software. Reading, Massachusetts: Addison-Wesley.

Wiegers K. E. 1996. Creating a Software Engineering Culture. New York: Dorset House Publishing.

Worrall J., Currie G., eds. 1978. The methodology of scientific research programmes. Cambridge, Massachusetts: Cambridge University Press.

Wurman R. S. 1991. Follow the Yellow Brick Road: Learning to Give, Take and Use Instructions. New York: Bantam Books.

Y

Yourdon E. 1997. Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects. Indianapolis, Indiana: Prentice Hall Computer Books.

Z

Zuse H. 1997. A Framework of Software Measurement. Walter de Gruyter, Inc.

Кем Кейнер, Джеймс Бах, Брет Петтикорд

Тестирование программного обеспечения: контекстно ориентированный подход

Перевел с английского С. Черников

Научный редактор А. Захарова

Руководитель дивизиона	*Ю. Сергиенко*
Руководитель проекта	*А. Питиримов*
Ведущий редактор	*Н. Гринчик*
Литературный редактор	*Н. Хлебина*
Художественный редактор	*В. Мостипан*
Корректоры	*Е. Павлович, Н. Терех*
Верстка	*К. Подольцева-Шабович*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 10.2024. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214,
тел./факс: 208 80 01.

Подписано в печать 23.08.24. Формат 70×100/16. Бумага офсетная. Усл. п. л. 28,380. Тираж 700. Заказ 0000.

КРОК

СОЗДАЕМ НАСТОЯЩЕЕ,
ИНТЕГРИРУЕМ БУДУЩЕЕ

croc.ru

КРОК — технологический партнер с комплексной экспертизой в области построения и развития инфраструктуры, внедрения информационных систем, разработки программных решений и сервисной поддержки.

Центры компетенций КРОК фокусируются на ключевых отраслевых кластерах — промышленность, финансовый сектор, розничные продажи, муниципальное управление, спорт и культура.

Ежегодно сотни проектов КРОК становятся системообразующими для экономики и социально-культурной сферы.
